

MICRO FOCUS

APS FOR z/OS

REFERENCE

Copyright © 2002 Micro Focus International Limited.
All rights reserved.

Micro Focus International Limited has made every effort to ensure that this book is correct and accurate, but reserves the right to make changes without notice at its sole discretion at any time. The software described in this document is supplied under a license and may be used or copied only in accordance with the terms of such license, and in particular any warranty of fitness of Micro Focus software products for any particular purpose is expressly excluded and in no event will Micro Focus be liable for any consequential loss.

Animator®, COBOL Workbench®, EnterpriseLink®, Mainframe Express®, Micro Focus®, Net Express®, REQL® and Revolve® are registered trademarks, and AAI™, Analyzer™, Application to Application Interface™, AddPack™, AppTrack™, AssetMiner™, CCI™, DataConnect™, Dialog System™, EuroSmart™, FixPack™, LEVEL II COBOL™, License Management Facility™, License Server™, Mainframe Access™, Mainframe Manager™, Micro Focus COBOL™, Object COBOL™, OpenESQL™, Personal COBOL™, Professional COBOL™, Server Express™, SmartFind™, SmartFind Plus™, SmartFix™, SourceConnect™, Toolbox™, WebSync™, and Xilerator™ are trademarks of Micro Focus International Limited. All other trademarks are the property of their respective owners.

No part of this publication, with the exception of the software product user documentation contained on a CD-ROM, may be copied, photocopied, reproduced, transmitted, transcribed, or reduced to any electronic medium or machine-readable form without prior written consent of Micro Focus International Limited.

Licenses may duplicate the software product user documentation contained on a CD-ROM, but only to the extent necessary to support the users authorized access to the software under the license agreement. Any reproduction of the documentation, regardless of whether the documentation is reproduced in whole or in part, must be accompanied by this copyright statement in its entirety, without modification.

U.S. GOVERNMENT RESTRICTED RIGHTS. It is acknowledged that the Software and the Documentation were developed at private expense, that no part is in the public domain, and that the Software and Documentation are Commercial Computer Software provided with RESTRICTED RIGHTS under Federal Acquisition Regulations and agency supplements to them. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFAR 252.227-7013 et. seq. or subparagraphs (c)(1) and (2) of the Commercial Computer Software Restricted Rights at FAR 52.227-19, as applicable. Contractor is Micro Focus, 9420 Key West Avenue, Rockville, Maryland 20850. Rights are reserved under copyright laws of the United States with respect to unpublished portions of the Software.

Table of Contents

++	11
01	11
RENAMES	13
88	14
Application Definition Report (AP01)	16
Application Field Edit Routines	18
Application Painter	24
Application Painter Member Processing Exits	27
Selection Code Processing	28
Application Reports	32
APSMACS Rule Library	37
AT END/INVALID KEY	39
ATTR	40
Attributes, Screen Fields	42
Bind and Translate Options, SQL	46
CA	49
CCODE	50
Checkin	51
Checkout	53
CIC-ADDRESS	55
CIC-ASSIGN	55
CIC-CANCEL	56
CIC-DELAY	57
CIC-DELETEQ-TD	58

CIC-DELETEQ-TS	59
CIC-FREEMAIN	59
CIC-GETMAIN	60
CIC-LOAD	61
CIC-READQ-TD	62
CIC-READQ-TS	63
CIC-RELEASE	64
CICS	65
CIC-SCHEDULE-PSB	65
CIC-SEND-TEXT	66
CIC-SERVICE-RELOAD	67
CIC-START	67
CIC-TERM-PSB	68
CIC-WRITEQ-TD	69
CIC-WRITEQ-TS	70
CLEAR	71
CLEAR-ATTRS	71
COBOL/2 Support	72
CODE	74
Comments	75
Component List (MS01)	76
CONTROL	78
Control Files	80
Control Points	88
Database Calls	97
Data Communication Calls	101
Data Structure Definition (DS01)	107
Data Structures	109

Date and Time Field Edits	113
DB/DC Target Combinations	118
DB-BIND	119
DB-CLOSE	120
DB-COMMIT	121
DB-DECLARE	122
DB-ERASE	128
DB-FETCH	134
DB-FREE	135
DB-GET	137
DB-MODIFY	138
DB-OBTAIN	143
DB-OPEN	158
DB-PROCESS	160
DB-ROLLBACK	174
DB-STORE	175
DB-SUBSCHEMA	180
DDIFILE Report (DB01)	181
DDI Statements	182
DDISYMB Flags	194
DECL	199
DLG-ISPEXEC	200
DLG-ISREDIT	201
DLG-SETMSG	201
DLG-VCOPY	203
DLG-VDEFINE	204
DLG-VDELETE	205
DLG-VREPLACE	206

DLG-VRESET	207
DPAR	207
DS	209
Entity Content Report (MS02).....	210
Entity Cross Reference (MD01)	212
Entity Parts List (EN01)	214
Entity Search Utility Report (GS01).....	216
Entity Use Report (EN02).....	219
ENTRY	221
Error Handling	222
Error Processing Messages	239
ESCAPE	241
EVALUATE	242
Exit Points.....	244
EXIT PROGRAM	245
Expressions, SQL.....	246
FD	247
Field Edits.....	248
Field Edit Values.....	251
Fields and Flags, Data Communication	256
Field/Screen Cross Reference (SC02)	262
FRFM	264
Functions, SQL	265
GENERATE	269
Generator Options.....	271
Generation Parameters, Screens.....	273
GROUP BY	278
GSAM Calls.....	280

ID Parameters:	281
IDM-COMMIT	282
IDM-CONNECT	283
IDM-DISCONNECT	284
IDM-IF	285
IDM-PROTOCOL	285
IDM-RETURN	287
IDM-ROLLBACK	288
IDMS	288
IDMS DB Sample Programs	289
IDMS Options	293
IF/ELSE-IF/ELSE	294
\$IM- Data Communication Calls	299
\$IM-FLD	301
\$IM-FSA	301
\$IM-POS	303
% INCLUDE	304
INITIATE	305
IO	305
ISPF Dialog Compatibility: with IMS DC, CICS	306
Job Control Cards	307
Joins	307
Keywords	311
Limits	319
LINK	321
LK	326
Macro/Program Cross Reference (MC01)	327
MFS Function Keys	329

MFS Trancode Construction	330
MID MOD Reorder	332
MOCK	333
MOCKUP LINES	334
Mock-Up Report (RP01)	335
Modified Data Tags, CICS Data Transmission	336
MSG-SW	337
NTRY	340
NULL Indicator Field	349
OCCURS	349
OPT	351
OVERPRINT	352
PAGE LIMIT	353
Panel Options, ISPF Dialog	355
PARA and Paragraphs	356
PERFORM	359
PF Key Values	361
Precompiler Options	364
PROC	370
Program Control Blocks, IO	371
Program DB/DC Report (PG02)	372
Program Definition Report (PG01)	374
Program Specification Blocks	375
Project and Group Options	376
REC	377
RED	378
REDEFINES	379
REFERENCE	380

REM	382
REPEAT	383
Report Mock-Ups	388
Report Sample Program and Mock-Up	389
Report Writer Structures	404
Reports, Application-Generated	407
Reserved Words	409
RESET-PFKEY	411
S-COBOL Structures	413
Scenario Definition Report (CN01)	419
Screen Hardcopy/Field Attribute Report (SC01)	420
Screen Redefinition	424
SCRNLIST	426
SD	429
SEARCH	430
SEND	431
SOURCE	438
Special Registers	440
SPNM	441
SQL	441
STOP RUN	444
STUB	444
Subselect Clause	445
SUM	447
SUPPRESS (IMS DB Option)	450
SUPPRESS (Report Writer)	450
SUPRA	451
SY* Keywords	452

System Service Calls	454
TERM	460
TERMINATE	461
TP-BACKOUT	462
TP-COMMAREA	463
TP-LINKAGE	468
TP-NULL	471
TP-PERFORM	472
TRUE/FALSE	473
TRUE, FALSE, ALWAYS, NEVER	475
TYPE	475
UNION	480
UNTIL/WHILE	483
USE BEFORE REPORTING	485
User Help	486
USERNAME	489
VALUE (Data Structure)	490
VALUE (Report Writer)	491
Values, Conversion Values, and Value Ranges	493
Variable Length File Support	494
WRITE ROUTINE	495
WS	496
XCTL	497

++

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Description: Generate a PANVALET ++INCLUDE statement.

Syntax: -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
++ PANVALETmembername

Comment: The preceding section keyword determines the placement of a data structure in the generated program. Associated section keywords are:

FD File Section (see *FD*)
SD Sort File Description (see *SD*)
WS Working-Storage Section (see *WS*)
LK Linkage Section (see *LK*)

Example: -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
WS
REC WS-INPUT-REC
 WS-IN-PART-NO N8
 WS-IN-DESC X50
 WS-IN-BASE-PRICE N6V2
01 WS-OUT-REC.
DS05 WSOUTREC
++ PANWSREC

01

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Description: Use the 01 keyword to:

- Define the input and output files in a batch program File Section.
- Define a data structure in Working-Storage or Linkage Section.
- Copy a data structure into the Working-Storage or Linkage Section.

- Define APS Report Writer statements for headings, footers, or detail lines.

Syntax: Format 1, define input or output files:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
IO      filename ASSIGN [TO] ...
        ORGANIZATION IS ...
01      input|outputrecordname          PIC clause
```

Format 2, define a data structure:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
01      COBOLdatastructure
        [05 COBOLdatastructure]
```

Format 3, copy a data structure:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
01      COBOLcopystatement
```

Format 4, define Report Writer line types:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
01      [dataname] TYPE [IS] reportgroup
        [NEXT GROUP [IS] number|PLUS number|NEXT PAGE][.]
        [LINE [NUMBER IS] number|PLUS number|NEXT PAGE]
        MOCKUP|M LINE|LINES linenumber1 [THRU linenumberN]
        [OVERPRINT] WHEN "characterstring" AT COLUMN column]
        [SOURCE] [IS] dataname [options]]
        [VALUE] [IS] literal]
        [REFERENCE [IS] dataname PIC[TURE] [IS] picclause
         [options]]
        [SUM|+ [IS] dataname [dataname] ...
         [UPON detailgroup [detailgroup] ...]
         [RESET [ON] [FINAL] dataname]
         [options]]
```

- Comments:**
- You must provide all necessary COBOL punctuation.
 - The following COPY statement formats are available for OS/VS COBOL. They are not supported by COBOL II, where the LANGLVL compiler option is set to 1.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
01      COPY copybookname

01      dataname COPY copybookname
```

```

01      dataname
        COPY copybookname

01      dataname COPY copybookname
        REPLACING fieldname1 BY fieldname2

```

- To write subordinate elementary data elements, include the level numbers in columns 12 - 72.
- If you use the COBOL/2 compiler, or your copybook contains an indexed table, enter the SWYS or SYLK keyword in the KYWD column and the APS statement % INCLUDE, rather than a COBOL COPY statement. % INCLUDE inserts the copybook into the program before precompilation, so the APS Precompiler can interpret the index. See the Customization Facility User's Guide for more information.
- The preceding section keyword determines the placement of a data structure in the generated program. Associated section keywords are

FD	File Section (see <i>FD</i>)
SD	Sort File Description (see <i>SD</i>)
WS	Working-Storage Section (see <i>WS</i>)
LK	Linkage Section (see <i>LK</i>)

RENAMES

Category: Data Structure Painter construct (see *Data Structures*)

Description: Code 66-level RENAMES clauses in your data structures.

Syntax: 66 *anytext*

Parameters: *anytext* Valid COBOL Syntax: for 66-levels.

Comment: A 66-level variable needs the same or deeper indentation than the data name it refers to.

Example: Data Structure Painter format:

```

-LINE- ----- DATA STRUCTURE PAINTER -----
000001     WORK-RECORD
000002         WORK-RECORD-GRP1
000003             WORK-FIELD-1     XX
000004                 WORK-FIELD-2     XXX
000005                     WORK-FIELD-3     PIC X(04)
000006         WORK-RECORD-GRP2
000007             WORK-FIELD-4     PIC X(02)
000008                 WORK-FIELD-5     PIC X(06)
000009         WORK-RECORD-GRP3
000010             WORK-FIELD-6     PIC X(08)
000011                 WORK-FIELD-7     PIC X(02)
000012     66 RECORD-1-REN RENAMES WORK-FIELD-1
000013     ... THRU WORK-FIELD-6

```

Generated COBOL code:

```

01 WORK-RECORD.
   05 WORK-RECORD-GRP1.
       10 WORK-FIELD-1           PIC XX.
       10 WORK-FIELD-2           PIC XXX.
       10 WORK-FIELD-3           PIC X(04).
   05 WORK-RECORD-GRP2.
       10 WORK-FIELD-4           PIC X(02).
       10 WORK-FIELD-5           PIC X(06).
   05 WORK-RECORD-GRP3.
       10 WORK-FIELD-6           PIC X(08).
       10 WORK-FIELD-7           PIC X(02).
   66 RECORD-1-REN RENAMES WORK-FIELD-1
                               THRU WORK-FIELD-6.

```

88

Category: Data Structure Painter construct (see *Data Structures*)

Description: Code 88-level clauses in your data structures.

Syntax:

```

88  dataname VALUE 'value1' [THRU 'value2']
...      'value3' [THRU 'value4']
...      'value5' [THRU 'value6']

```

```

.
.
...          'valueN' [THRU 'valueN']

```

Parameters: *value* 88-level value; do not enclose numeric values in single quotation marks.

- Comments:**
- An 88-level variable needs the same or deeper indentation as the data name it refers to.
 - You can code the VALUE clause either
 - On the same line as *dataname* if it fits entirely on that line.
 - On its own line and continue it on subsequent lines with the continuation symbol.

Examples:

- Data Structure Painter format:

```

-LINE- ----- DATA STRUCTURE PAINTER -----
000001      WRK1-FIELD-1 X5
000002      88 OPEN-VAL
000003      ... V 'OPEN' 'BEGIN'
000004      ... 'START'
000005      88 OPEN-FLAG
000006      ... VALUE 'ON'
000007      ... 'YES'

```

Generated COBOL code:

```

01  WRK1-FIELD-1          PIC X(5).
    88  OPEN-VAL          VALUE 'OPEN' 'BEGIN'
                                'START'.
    88  OPEN-FLAG          VALUE 'ON'
                                'YES'.

```

- Data Structure Painter format:

```

-LINE- ----- DATA STRUCTURE PAINTER -----
000009      WRK1-FIELD-2 X(04)
000010      88 SUB-LVL1
000011      ... V 'ABCD' 'EFGH'
000012      ... 'FFFF' THRU 'MMMM'
000013      ... 'PPPP' THRU 'ZZZZ'
000014      88 SUB-LVL2
000015      ... V LOW-VALUES

```

Generated COBOL code:

```

01  WRK1-FIELD-2          PIC X(04).
    88  SUB-LVL1          VALUE 'ABCD' 'EFGH'
                                'FFFF' THRU 'MMMM'
                                'PPPP' THRU 'ZZZZ'.
    88  SUB-LVL2          VALUE LOW-VALUES.

```

- Data Structure Painter format:

```

-LINE- ----- DATA STRUCTURE PAINTER -----
000016          WRK1-FIELD-3 H
000017          88 SUB-LVL3
000018          ... V -1000 +1000
000019          ... -900 THRU -700
000020          ... +500 THRU +800

```

Generated COBOL code:

```

01  WRK1-FIELD-3          PIC S9(4) COMP.
    88  SUB-LVL3          VALUE -1000 +1000
                                -900 THRU -700
                                +500 THRU +800.

```

Application Definition Report (AP01)

Category: APS-generated report (see *Application Reports*)

Description The Application Definition Report displays each component of an application in a separate report section.

This report consists of the following collection of reports.

- Data Structure Definition Report (DS01) (*Data Structure Definition (DS01)*)
- Program Definition Report (PG01) (*Program Definition Report (PG01)*)
- Report Definition Report (RP01)
- Screen Hardcopy/Field Attribute Report (SC01) (*Screen Hardcopy/Field Attribute Report (SC01)*)

Application Field Edit Routines

Category: Screen Painter feature (see *Field Edits*)

Description: Specify additional edits or tests for input or output data.

Procedure: To assign an edit routine, follow these steps.

- 1 From the Screen Painter, access the Field Edit Facility.
- 2 To access the Application Edits screen, select the Application Editing prompt on any Field Edit screen.
- 3 Complete the following options.

Option	Description
Type	Indicate whether this application edit is a paragraph, subprogram, or APS macro. Default is P(agraph).
Name	Enter a descriptive name for the application edit; maximum 32 characters.
Arguments	To pass a screen field or error flag, prefix the name with the screen name and a hyphen. If the field is located in a list box or combination box, APS suffixes the name with (APS-ROW-SUB). Enter the following arguments separated by commas and enclose literals in single quotation marks.
Paragraph	Data names or literals that pass to the paragraph through a PERFORM with arguments statement in the generated program.
Subprogram	Data names that appear on the CALL USING statement in the generated program.
APS macros	Customization Facility macro terms, literals, and numeric literals. Do not enclose arguments with double quotation marks.

Option	Description
Execute Before/After APS Edits	Specify when the program executes this application edit--before or after the normal APS field edit routine. Default is b(efore). See also "Comments:" below.
Paragraph COPYLIB or APS Macro USERMACS Member	Specify an associated COPYLIB member or a paragraph or the associated USERMACS member name for an APS macro.
Working-Storage COPYLIB Member	Specify a Working-Storage COPYLIB member to be included in the program Working-Storage section.

Alternately, for screen fields, you can select a predefined edit from the Application Edit List, which is a centralized collection of routines maintained by your APS Administrator. To do so:

- 1 On the Application Editing screen, enter **applist listname** in the **Command** field, where listname is the name of the list of available edits. See your APS Administrator for the name of the list at your site. The Application Selection screen displays.

```

COMMAND ==>                                     SCROLL==> PAG
SCREEN FIELD NAME : CUSTOMER-NO          LEN 006 ROW 009 COL 027
NAME          TYPE  DESCRIPTION
CUSTOMER-NAME I    CHECK FOR DUPLICATE CUSTOMER NAMES
s CUSTOMER-VERIFY I  ENSURE CUSTOMER NUMBER IS VALID
*****
***** BOTTOM OF DATA *****
    
```

- 2 Type s before the input or output edit routine you want.
- 3 Press Enter to select the edit and return to the Application Editing screen. APS copies the field values from the selected edit routine, overlaying existing entries. You can modify the selected edit routine on this screen.

Comments: • You can interrogate the following APS-generated flags and data fields to determine input data errors.

APS-EDITS-PASSED	Flag that APS sets to T(rue) if there are no input errors and F(alse) if there are input errors
------------------	---

APS-EDIT-ERRORS-CTR	Data field containing the number of screen fields in error
<i>screenname-fieldname</i> -FLAG	Flag that APS sets to T(ue) if an error occurs and F(alse) if not

- You can use the following APS-generated fields to code your application edit routines. The option Execute Before/After APS Edits determines which field to reference, as shown in the table below.

Representation	Execute Option	Name and Format
Input	After APS edits	<i>Screenname-fieldname</i> . Internal picture was updated by any previous field edits.
Input	Before APS edits	<i>Screenname-fieldname</i> -INPT. Data field containing any invalid data entered by the end user. Edits have not been performed; data is in screen input format. Field length is field length painted on the screen; definition is PIC X.
Output	After APS Edits	<i>Screenname-fieldname</i> -EDIT. Data was moved to -EDIT by previous output edits. Data may contain special symbols. Default length is field length painted on the screen; definition is PIC X.
Output	Before APS Edits	<i>Screenname-fieldname</i> . Data was not processed by previous output edits and is still in the internal format. Data in this field moves to the -EDIT field when an output format is specified. Default length is field length painted on the screen; definition is PIC X.

Representation	Execute Option	Name and Format
Input or Output	Before or After APS Edits	(APS-ROW-SUB). This data field contains the row number being processed for fields in a repeated block, or in a list box or combination box.
Input or Output	Before or After APS Edits	APS-EDIT-MESSAGE. Data field that lets you display an error message in addition to any you defined in the field edits for the field. Move a literal or data name containing the text into this field.
Input or Output	Before or After APS Edits	<i>Screenname-fieldname-LEN</i> . For CICS only. Contains the length of the field; definition is PIC X.
Input or Output	Before or After APS Edits	<i>Screenname-fieldname-ATTR</i> . For CICS only. Contains the attribute values assigned to the field; definition is PIC X.

- APS processes input data from the *screenname-fieldname-INPT* field. After the data passes the input edits, the data moves to *screenname-fieldname* and takes on the characteristics of the internal picture. For output, the data moves to the *screenname-fieldname-EDIT* field and takes on the characteristics of the output format specifications.
- Application edit routines execute for a field whether the data passes or fails field edits, unless you set various switches in the APFEIN control file (see *Control Files*). To indicate an error in an application edit routine, move T to *screenname-fieldname-FLAG*. This tells APS to execute error processing; you can display an additional error message from the APS-EDIT-MESSAGE field.

Examples: The following is sample generated code for a screen record with input and output edits.

```

01  EMPLOYEE-RECORD.
    05  EMPLOYEE-SALARY-GRP .
        10  EMPLOYEE-SALARY-INPT.
            15  FILLER                                PIC X(03) .
            15  EMPLOYEE-SALARY                        PIC S9(05)V99 .
        10  EMPLOYEE-SALARY-OUTP REDEFINES
                                                    EMPLOYEE-SALARY-INPT .
            15  EMPLOYEE-SALARY-EDIT                    PIC $$$,$$9.99 .

```

The following three examples show the generated code when you code an application edit routine as a paragraph, subprogram, and macro. Each routine:

- Validates the part number entered in the CUSTOMER-NO field.
- Is invoked during field edit input logic execution.
- Executes After APS Edits.
- Passes the following arguments
*CUSTOMER-NO, *CUST-NO-FLAG, APS-EDIT-MESSAGE

In the first example illustrated below, the edit routine uses the VALIDATE-CUSTOMER-NO paragraph, which resides in the CUSTPARA copylib, as shown below.

```

COMMAND ==> _

SCREEN FIELD NAME : CUSTOMER-NO      LEN 006  ROW 010  COL 024
  OUTPUT Application Edit Type ==> p
    P - Paragraph      S - Subprogram      M - APS Macro
  Name ==> validate-customer-no
  Arguments:
    ==> *customer-no,*cust-no-flag,aps-edit-message
    ==>
    ==>
  Execute before/after APS edits ==> a  [ B - Before, A - After ]
  Paragraph COPYLIB or APS Macro USERMACS member (optional):
    Member Name ==> custpara
  WORKING-STORAGE COPYLIB member ==>  (optional)

```

The VALIDATE-CUSTOMER-NO paragraph receives the SCRNCUSTOMER-NO data, verifies it exists, and returns a T(rue) value to SCRNCUST-NO-FLAG if the part number is not found. Then, APS checks

the SCRN-CUST-NO-FLAG to determine whether an error was found. If so, the APS-EDIT-MESSAGE text displays on the screen.

```
PERFORM VALIDATE-CUSTOMER-NO( SCRN-CUSTOMER-NO,
... SCRN-CUST-NO-FLAG, APS-EDIT-MESSAGE)
.
.
.
VALIDATE-CUSTOMER-NO(+WS-CUSTOMER-NO,-WS-ERROR-FLAG,
... WS-MESSAGE)
  DB-OBTAIN REF CUST-RECORD
  ... WHERE CUSTOMER-NO-KEY = WS-CUSTOMER-NO
  IF NTF-ON-REC
    MOVE 'T' TO WS-ERROR-FLAG
    MOVE 'CUSTOMER NUMBER NOT FOUND ON FILE'
    ... TO WS-MESSAGE
    MOVE ' ' TO WS-ERROR-FLAG
  ELSE
    MOVE 'T' TO WS-ERROR-FLAG
    MOVE 'ABNORMAL RETURN ON PART NBR VALIDATION'
    ... TO WS-MESSAGE
```

In the second example, the edit routine uses the CUSTVER subprogram. The subprogram accepts the arguments into the Linkage Section.

```
CALL 'CUSTVER' USING
... SCRN-CUSTOMER-NO,
... SCRN-CUST-NO-FLAG,
... APS-EDIT-MESSAGE
```

In the third example, the edit routine uses the \$VALIDATE-CUSTOMER-NO macro, which resides in USERMACS CUSTMAC. The \$VALIDATE-CUSTOMER-NO macro receives the SCRN-CUSTOMER-NO as an argument, verifies it exists, and returns a T(rue) value to SCRN-CUST-NO-FLAG if the part number is not found. Then, APS checks the SCRN-CUST-NO-FLAG to determine whether an error was found. If so, the APS-EDIT-MESSAGE text displays on the screen.

```
$VALIDATE-CUSTOMER-NO("SCRN-CUSTOMER-NO",
% ... "SCRN-CUST-NO-FLAG",
% ... "APS-EDIT-MESSAGE")
.
.
% DEFINE $VALIDATE-CUSTOMER-NO( &CUSTOMER-NO, &ERROR-FLAG,
% ... &MESSAGE )
  $DB-OBTAIN("REF CUST-RECORD",
  % ... <%&CQ WHERE CUSTOMER-NO-KEY = &CUSTOMER-NO&CQ>)
```

```

IF NTF-ON-REC
    MOVE 'T' TO &ERROR-FLAG
    MOVE 'PART NUMBER NOT FOUND ON FILE' TO &MESSAGE
ELSE-IF OK-ON-REC
    MOVE ' ' TO &ERROR-FLAG
ELSE
    MOVE 'T' TO &ERROR-FLAG
    MOVE 'ABNORMAL RETURN ON CUSTOMER NUMBER VALIDATION'
    ...TO &MESSAGE

```

Application Painter

Description: Create an application definition by listing all its components in a matrix that indicates their relationship to each other. Specify requirements on the Application Painter screen, as follows:

Field	Description and Values
DC	<p>Data communications target. Valid targets are CICS, DLG (ISPF Dialog), IMS, ISPF, MVS (batch). Specify the target as follows:</p> <p>If application contains... Specify this DC target...</p> <p>Only online programs Your online DC target.</p> <p>Only batch programs MVS. In addition, leave each program Screens field blank.</p> <p>Both online and batch programs Your online DC target. To identify the batch programs, enter *batch in the Screens field next to each batch program name.</p> <p>For a list of valid DB/DC combinations for generating executable programs, see <i>DB/DC Target Combinations</i>.</p>
DB	<p>Database target. Valid values are DLI (or IMS), IDMS, VSAM.</p> <p>To specify a SQL target, leave the DB field blank or let default to VSAM. Then go to the Generator Options screen and specify the SQL target.</p> <p>If your application accesses multiple database targets, specify the DB target as follows.</p> <p>If application accesses . . . Specify this DB target . . .</p> <p>.</p> <p>Two DB targets, including VSAM The non-VSAM target--APS always gives you access to the VSAM target.</p>

Field Description and Values

Field	<p>Two or more DB targets, excluding VSAM</p> <p>Any DB targets. When you generate the programs, first generate just the programs of your specified DB target. Then change the DB target to the next target and generate just the programs of that next target. For example, if your application accesses both SQL and IMS subschemas, generate your SQL programs separately from your IMS programs.</p> <p>For a list of valid DB/DC combinations for generating executable programs to run on various operating systems, see <i>See DB/DC Target Combinations</i>.</p>															
Screen Size	<p>Specify the size of the screen for your application by selecting one of the following application screen sizes from the Screen Size field.</p> <table border="0" style="width: 100%;"> <thead> <tr> <th style="text-align: left; padding-right: 20px;">Application Screen Size</th> <th style="text-align: left; padding-right: 20px;">Dimension</th> <th style="text-align: left;">Development Screen Size</th> </tr> </thead> <tbody> <tr> <td>MOD2</td> <td>24 x 80</td> <td>MOD2, MOD3, MOD4, or MOD5</td> </tr> <tr> <td>MOD3</td> <td>32 x 80</td> <td>MOD3 or MOD4</td> </tr> <tr> <td>MOD4</td> <td>43 x 80</td> <td>MOD4</td> </tr> <tr> <td>MOD5</td> <td>27 x 132</td> <td>MOD5</td> </tr> </tbody> </table>	Application Screen Size	Dimension	Development Screen Size	MOD2	24 x 80	MOD2, MOD3, MOD4, or MOD5	MOD3	32 x 80	MOD3 or MOD4	MOD4	43 x 80	MOD4	MOD5	27 x 132	MOD5
Application Screen Size	Dimension	Development Screen Size														
MOD2	24 x 80	MOD2, MOD3, MOD4, or MOD5														
MOD3	32 x 80	MOD3 or MOD4														
MOD4	43 x 80	MOD4														
MOD5	27 x 132	MOD5														
Program	<p>Enter program names; maximum eight characters. The first character must be alphabetic; others can be alphabetic, numeric, or the special characters #, \$, or @. The names all and dummy are invalid.</p>															
Screen	<p>Associated screen name; eight-character maximum, except for IMS DC and ISPF Prototyping, which have a seven-character maximum. The first character must be alphabetic, others can be alphanumeric. For batch programs, enter *batch in the Screens field, on the same row as the program name. For online programs, enter the program associated screen name, on the same row as the program name.</p>															
IO	<p>Specify whether the screen is input-only (i), output-only (o), or input/output (io). For batch programs, leave the IO field blank.</p>															
Report	<p>Batch program report mock-up name; eight character maximum. The first character must be alphabetic or the special characters #, \$, or @; others can be any of these or numeric.</p>															
Data St	<p>Name of any data structure file that the program will reference; eight character maximum. The first character must be alphabetic; others can be alphanumeric. If the program references multiple data structure files, enter their names on subsequent rows. To make the data structures global, or available to all programs of the application, enter their names on rows above all programs.</p>															
Ty	<p>Type of data structure file, indicating the program location where you plan to include it, as follows.</p>															

Field	Description and Values
	WS Working-Storage Section
	LK Linkage Section
	CA Commarea
Schema	Subschema or PSB name; eight character maximum. The first character must be alphabetic; others can be alphanumeric. To make the subschema or PSB global, or available to all programs of the application, enter its name on a row above all programs.
User Mac	Name of user-defined macro library file that the program will reference; eight characters maximum. The first character must be alphabetic; others can be alphanumeric. If the program references multiple macro library files, enter their names on subsequent rows. To make the files global, or available to all programs of the application, enter their names on rows above all programs. The files must reside in your <i>project.group</i> USERMACS PDS.
Loc	Location of the macro library file, indicating the program location where you plan to invoke its macros, as follows:
	T Default; top of program, before Identification Division
	B Bottom of program
	WT Top of Working-Storage Section
	WS Working-Storage Section, after any data structures you include in the Data Str field
	WB Bottom of Working-Storage Section
	LT Top of Linkage Section
	LK Linkage Section, after any data structures you include in the Data Str field
	LB Bottom of Linkage Section
	IO Top of Input-Output Section
	FD Top of File Section
	RP Top of Report Section
	CA Top of Commarea
Comments:	<ul style="list-style-type: none"> • You can include procedural subroutines that any program of the application can reference, known as global stubs. To do so, enter the stub name in the Program field, and enter *stub in the field. Regardless of the row where you enter a global stub name, any program of the application can reference it. See <i>STUB</i>.

- To create a new application definition quickly, you can copy an existing one and modify it. To do so, use the Create Like function on the Painter Menu.
- If you are creating a CICS or IMS DC application that accesses SQL or VSAM databases, and you want to create an application prototype to execute and test within the APS Prototype Execution facility, set the DC target to ISPF and the DB target to SQL or VSAM. After testing the ISPF prototype, change the DB/DC targets to the production targets, and regenerate the application.
- Deleting a component from the Application Painter matrix removes it from the application definition, but not from the APS Repository; the component is available to add to other applications. However, if you delete a component from the Painter Menu, APS removes it from the APS Repository; you must delete it from all applications that reference it.
- If you are creating an IMS application that does not access the IMS message queue, specify MVS as your DC target.

Application Painter Member Processing Exits

Description On the Application Painter, the selection fields next to programs, screens, reports, and data structures accept the following APS-defined selection codes.

This code...	Represents...
ox	Select Online Express program
s, e	Select or Edit
b	Browse
g	Generate
r	Report
bd	BIND

You can create selection codes to do additional things by writing a member processing exit subroutine. For example, you could create a member processing exit that contains logic to archive an entity to tape when the user-defined selection code T is entered. You can also create

member processing exits to modify or disable the APS-defined selection codes.

Procedure: To write an Application Painter member processing exit, follow these steps:

- 1 Copy the APS-provided member processing exit program template, A1UXAP01, which resides in the APSPROG PDS. Modify the copy with your own logic to suit your needs, and generate the program. Note that in the A1UXAP01 program, a like-named macro library is included; the \$APS-UXAP01-LINK-PARMS macro member of this library defines the parameters you use in the exit program.
- 2 Access the APS Administration Configuration screen and define your member processing exit name.
- 3 Test your member processing exit. If you get a system abend code 806 from the Application Painter, the exit is not in the system search path - perhaps you entered the wrong name in step 2, or your exit did not compile.
- 4 To make your exit available from any Project and Group, copy the generated load module to an appropriate load library, such as the APS software library ISPLLIB2, or any library in the system search path.

Selection Code Processing

When you enter one or more selection codes on the Application Painter, the Painter first validates that the selection codes were entered correctly. Then, if no errors are found, the Painter processes the selection codes.

You can code logic in an exit program for any of three Application Painter processing points - during validation, before processing, and after processing. APS provides the following two structures in the exit macro and the exit program template respectively, to determine which function is requested.

```
P-FUNC-CD          X(1)
  88 P-FUNC-CD-VALIDATE      V 'V'
  88 P-FUNC-CD-PRE-PROCESS  V 'B'
  88 P-FUNC-CD-POST-PROCESS V 'A'
```

```

      .
      .
EVALUATE P-FUNC-CD
WHEN 'V'
    PERFORM VALIDATE-FUNC
WHEN 'B'
    PERFORM CUSTOM-PREPROCESS
WHEN 'A'
    PERFORM CUSTOM-POSTPROCESS

```

APS also provides the following variables in the exit macro. To define your own selection code(s) to the exit program, reference these variables in your program.

```

% &APS-UE-SEL-BROWSE           = 'B'
% &APS-UE-SEL-BIND            = 'BD'
% &APS-UE-SEL-EDIT            = 'E'
% &APS-UE-SEL-GEN             = 'G'
% &APS-UE-SEL-ONLINE-EXPRESS  = 'OX'
% &APS-UE-SEL-REPORT          = 'R'
% &APS-UE-SEL-SELECT          = 'S'

```

Validation

The Painter validates each selection code entered. It also calls the exit program's validation paragraph once per selection code. In the paragraph, you can use the following return codes.

Return Code	Meaning
&APS-UE-RC-OK	All validations for the selection code are successful; the exit program should handle this selection and bypass APS processing.
&APS-UE-RC-UNKNOWN-SEL-CD	The selection code is not user-defined; APS should handle it.
&APS-UE-RC-CONTINUE	The selection code is a standard APS selection code. Both the exit program and APS should process this selection code.
&APS-UE-RC-USER-ERROR	The selection code is in error; APS should stop validation and redisplay the Application Painter screen for user correction.

For example:

```

/* *****
/* USER LOGIC TO VALIDATE SELECTION CODES
/* *****
PARA VALIDATE-FUNC
/* VALIDATE THE SELECTION CODE...
IF P-SEL-CD = 'yourcode'
    RETURN-CODE = &UE-RC-OK
ELSE
    RETURN-CODE = &UE-RC-UNKNOWN-SEL-CD

```

Preprocessing:

If validation was successful, the Application Painter processes the selection codes in this order:

g	Generate screens, then programs
bd	Bind
r	Report
ox	Online Express
s, e, b, and user-defined codes	Codes processed as they appear on Application Painter, from top to bottom, and right to left

For each selection code, the Application Painter checks the return code set by the exit program during validation. If the return code is &APS-UE-RC-OK or &APS-UE-RC-CONTINUE, the Application Painter calls the exit program preprocessing paragraph; otherwise standard Application Painter processing occurs.

In the exit program preprocessing paragraph, write your custom preprocessing logic, and then set one of these return codes.

Return Code	Meaning
&APS-UE-RC-OK	Processing of this selection code is complete.
&APS-UE-RC-CONTINUE	The exit program preprocess paragraph is successful; continue with standard Application Painter processing.

Return Code	Meaning
&APS-UE-RC-POSTPROCESS	The exit program preprocess paragraph is successful; continue with both standard Application Painter and exit program postprocessing.
&APS-UE-RC-USER-ERROR	The exit program preprocess paragraph is unsuccessful; continue processing with the next selection code.

For each selection code, when the exit program preprocess paragraph returns &APS-UE-RC-CONTINUE or &APS-UE-RC-POSTPROCESS, the Application Painter attempts to perform its standard processing. Then, for &APS-UE-RC-POSTPROCESS, the Application Painter calls the exit program postprocess paragraph. The call parameter P-APS-STATUS indicates the success or failure of the standard Application Painter processing.

Postprocessing:

Use the exit program postprocessing paragraph to execute any action after the standard Application Painter processing. A common action is to free resources allocated by the preprocess paragraph. The postprocessing paragraph should set one of these return codes.

Return Code	Meaning
&APS-UE-RC-OK	The exit program postprocessing paragraph is successful.
&APS-UE-RC-ERROR	The exit program postprocessing paragraph failed.

- Comments:**
- A member processing exit can display one or more ISPF screens.
 - A member processing exit can invoke one or more subroutines.
 - Each member processing exit must return to the Application Painter.
 - A member processing exit cannot directly or indirectly invoke the Application Painter.
 - A member processing exit can set a message using the SETMSG paragraph provided in A1UXAP01. Depending on the circumstances, a subsequent APS message can override this message.

Application Reports

Description: APS provides a set of reports that help you understand your application and its various components. Use these reports as you develop an application to determine the status of your work and the tasks left to complete. Some reports help you to troubleshoot problems in an application that you are developing, or to determine the impact of a proposed change. Others help you to verify the results of your work. Once you have fully implemented an application, use the APS reports to document it so that developers who later maintain or enhance the application can easily understand it in detail.

You can produce reports on an entire application, on selected components, or on selected members of components. You can produce reports from the Report Generator, Painter Menu, Application Painter, or Documentation Facility, as follows.

Report	Available from
Application Definition (AP01) lists and describes all components of an application except the scenario prototype. See <i>Application Definition Report (AP01)</i> .	Painter Menu Application Painter Report Generator
Component List (MS01) catalogs and totals the components for each painter. See <i>Component List (MS01)</i> .	Documentation Facility
Data Structure Definition (DS01) lists and describes structures that you create in the Data Structure Painter. See <i>Data Structure Definition (DS01)</i> .	Painter Menu Application Painter Report Generator
DDIFILE (DB01) describes the contents of the file that contains information about your database, formatted to APS specifications. See <i>DDIFILE Report (DB01)</i> .	Documentation Facility
Entity Content (MS02) lists summary information for each application component. See <i>Entity Content Report (MS02)</i> .	Documentation Facility

Report	Available from
Entity Cross Reference (MD01) cross references and totals application components. See <i>Entity Cross Reference (MD01)</i> .	Documentation Facility
Entity Parts List (EN01) catalogs selected parts of one or more application components. See <i>Entity Parts List (EN01)</i> .	Documentation Facility
Entity Search Utility (GS01) lets you create reports on application components that meet the selection criteria that you specify. See <i>Entity Search Utility Report (GS01)</i> .	Documentation Facility
Entity Use (EN02) lists components that copy, include, or otherwise use the target component. See <i>Entity Use Report (EN02)</i> .	Documentation Facility
Field/Screen Cross Reference (SC02) lists application screens along with their I/O and text fields. See <i>Field/Screen Cross Reference (SC02)</i> .	Documentation Facility
Macro/Program Cross-Reference (MC01) lists macros and the programs that use them. See <i>Macro/Program Cross Reference (MC01)</i> .	Documentation Facility
Mock-Up (RP01) lists and displays report mock-ups as painted in the Report Mock-Up Painter. See <i>Mock-Up Report (RP01)</i> .	Painter Menu Application Painter Report Generator
Program DB/DC (PG02) lists the screens and the subschemas or PSBs used by a program. See <i>Program DB/DC Report (PG02)</i> .	Documentation Facility
Program Definition (PG01) provides a printout of programs created in APS. See <i>Program Definition Report (PG01)</i> .	Painter Menu Application Painter Report Generator
Scenario Definition (CN01) describes components created in the Scenario Prototype Painter. See <i>Scenario Definition Report (CN01)</i> .	Painter Menu Application Painter Report Generator

Report	Available from
Screen Hardcopy/Field Attribute (SC01) displays the components of a screen as painted in the Screen Painter as well as field attribute and field edit information. See <i>Screen Hardcopy/Field Attribute Report (SC01)</i> .	Painter Menu Application Painter Report Generator

Procedures: Use the following procedures to produce APS reports from the Report Generator, Painter Menu, Application Painter, and Documentation Facility.

Report Generator

Produce reports from the Report Generator following these steps.

- 1 On the Painter Menu, enter in the **Type** field the component type that you want to report on - **ap**(plication), **cn** (scenario), **ds** (data structure), **pg** (program), **rp** (report mock-up), or **sc**(reen).
- 2 Leave the **Member** field blank.
- 3 Enter **report** in the **Command** field and press Enter to display the Report Generator screen.
- 4 Enter one of the following selection criteria.
 - To report on all members of all component types, type **1** in the **Option** field. Make sure that all entry fields are blank on the screen.
 - To report on all members of the one component type, type **2** in the **Option** field. Leave the value that displays in the **Library** field.
 - To report on a specific member of a component, type **3** in the **Option** field. Leave the value that displays in the **Library** field, and enter the component's member name in the **Member Name** field.
 - To report on a range of members in a component, type **3** in the **Option** field. Enter the component type in the **Library** field and the value range of the members you want to report on in the **Range Greater** and **Range Less** fields.
- 5 Press Enter to submit a job to produce the report.

Painter Menu

Produce reports from the Painter Menu following these steps.

- 1 In the **Type** field, enter the component type that you want to report on - **ap**(plication), **cn** (scenario), **ds** (data structure), **pg** (program), **rp** (report mock-up), or **sc**(reen).
- 2 In the **Member** field, enter the member name that you want to report on.
- 3 Enter **report** in the **Command** field and press Enter. The Print Report screen displays.
- 4 Press Enter to submit a job to generate the report.

Application Painter

Produce reports from the Application Painter following these steps.

- 1 To report on all members of all components, or all members of a specific component, do one of the following.
 - Type **report** in the **Command** field to report on all members of all components of an application.
 - Enter **report componenttype all** in the **Command** field to report on all members of a specific component. *Componenttype* can be **ap**(plication), **cn** (scenario), **ds** (data structure), **pg** (program), **rp** (report mock-up), or **sc**(reen).
- 2 To report on a specific member of a component, do one of the following.
 - Enter **report componenttype componentname** in the **Command** field.
 - Type **r** next to the component name in the Application Painter matrix.
- 3 Press Enter to submit a job to generate the report.

Documentation Facility

Produce reports from the Documentation Facility following these steps.

- 1 From the APS Main Menu screen, enter option **2** in the **Option** field. The Dictionary Services screen displays.

- 2 From the Dictionary Services screen, enter option **3** in the **Option** field. The Documentation Facility screen displays.
- 3 Select the desired report from the Actions listing on the action bar or enter the applicable option number in the **Option** field. A selection criteria screen displays for the selected report.
- 4 Enter any selection criteria. See the individual report descriptions for details.
- 5 Press Enter to submit a job to generate the report.

- Comments:**
- Although the procedures for producing reports from the Painter Menu and the Report Generator are very similar, the Report Generator offers more flexibility. On the Painter Menu you can report on specific component members, while the Report Generator lets you report on any range of component members that you specify.
 - Make sure that the current Project and Group contain the application entities you want to report on. If you want to change your Project\Group, do the following. Alternatively, from the APS Main Menu, select option 0, Options. The APS Options Menu displays. Select option 2, Project Group Environment. Enter the desired Project and Group values and press PF3 to return to the APS Main Menu.
 - Set up job card information so that you can submit jobs to generate reports. To do so, from the APS Main Menu, select option 0 and then option 6, Job Card Options. Alternatively, enter **opt** in the **Command** field.
 - To set the job class, from the APS Main Menu, select option 0 to invoke the Generator Options panel. Then select option 1 and reference the job card you created.

APSMACS Rule Library

Description: Contains macros used by APS during generation. Place user macros in *project.group.USERMACS*.

Note: This feature is closely associated with the Control Files feature. See *Control Files*.

Category: APS library.

Rule Name	Usage
A1CMLIB	Endevor Configuration Management
A1UXAP01	Application Painter User Exit
APAUTOED	Character based Automatic Edits
APBTCHTP	Batch Generator
APCICSTP	CICS Generator
APCMSCAN	TCP/IP Generator
APDB2DB	SQL Generator (all SQL database types)
APDLGMAC, APDLGTP	ISPF Dialog Generator
APFEDCLR, APFEMACS	Field Edits
APHLPMAC	Online Help
APIDMSPT	IDMS Pass Thru Generator
APIDMSTP	IDMS DC Generator
APIMSDB	IMS DB Generator
APIMSTP	IMS/TM Generator
APMSGADB	Database Generator Errors (all database types)
APMSGBTC	Batch Generator Messages
APMSGCIC	CICS Messages
APMSGDB2	SQL Generator Messages (all SQL database types)
APMSGDCL	IMS Database Messages
APMSGDLG	DLG Messages

Rule Name	Usage
APMSGHLP	User Help Messages
APMSGIDB, APMSGIDC	IDMS Generator Messages
APMSGIMS	IMS Generator Messages
APMSGMDC	Character based TP messages (CICS, IMS)
APMSGVSM	VSAM Generator Messages
APPCVSM	VSAM Generator
APPIPMAC	TCP/IP Generator
APSBASE	APS Base (all targets)
APSCRGEN	Character based Screen Generator
APSDBCMD	Database Generator (all database types)
APSUBSCH	Subschema Processor
APVBXMAC	Visual Basic Extension (VBX) Generator
APVSMCIC	VSAM Generator (CICS only)
APVSMDB	VSAM Generator (all platforms)
APVSMMVS	VSAM Generator (batch)
DCLMACS	IMS DB and VSAM Generator
DDIAMS	VSAM IDCAMS Generator (mainframe only)
DXPG02	PG02 (Program Painter) Report Generator
IDMDBMAC	IDMS DB Generator
IDMPROTO	IDMS Messages
IMSDBMAC, IMSPHYS	IMS DB Generator
SSMXPG02	PG02 (Specification Editor) Report Generator
VSMDBMAC	VSAM Generator
VSMPHYS	VSAM Generator (all platforms)

ATTR

Category: Data communications call (see *Data Communication Calls*)

Description: Modify the screen I/O field attributes attributes at run time. To reset all screen field attributes to their original painted values, see *CLEAR-ATTRS*.

Syntax: *ISPF Prototyping*

```
[TP-]ATTR screenname POS fieldname[(subscript)]
```

CICS, IMS DC, and ISPF Dialog

```
[TP-]ATTR screenname
... attribute1[+attribute2...]
... fieldname[(subscript)][+fieldname[(subscript)] ...]
```

Keywords	<i>attribute</i>	Specify one or more attributes as follows.
	BRT	Bright character images
	NORM	Normal character images
	DARK	Suppress character display
	MDTON	Enable modified data tags (CICS only)
	MDOFF	Disable modified data tags (CICS only)
	NUM[LOCK]	Enable numeric locking (not applicable for DDS)
	NOLOCK NUMOFF	Disable numeric locking (not applicable for DDS)
	POS[ITION]	Position the cursor at the first field in the string (not applicable for DDS)
	PROT	Specify write-protection
	ASKIP	Specify write-protection (not applicable for DDS)
	UNPROT NO PROT	Cancel write-protection
	COLSEP	Specify column separator (DDS only)

DET Enable light pen detection
 NODET|DET Disable light pen detection
 OFF

Color Specify one of the following character image colors:

BLUE | BL
 GREEN | GN
 PINK | PK
 TURQ | TQ
 DEFCOL (default)
 NEUTRAL | NU
 RED | RD
 YELLOW | YL

Highlighting Specify one of the following character highlights:

BLINK Blinking cursor
 NOBLINK Nonblinking cursor
 RVID Reverse video
 NORVID Normal video
 UNDER Underlining
 NOUNDER No underlining

fieldname Screen field name(s). Code only the field name, not its
 [*(subscript)*] screen name prefix; code the field subscript if applicable.

screenname Screen name; value must be literal (maximum 8 characters).

- Comments:**
- When redefining a screen, use the original name.
 - Code only as many fieldnames as can fit on a single line. To code more field names, code another ATTR.
 - For character-based applications, to modify color or highlighting, set the **Extattr Modifiable** field to **Y** in the Screen Painter Screen Generation Parameters: *screen*.
 - You can code attributes that are not supported, such as when prototyping, without causing an error. They are ignored at generation/execution time.

- Examples:**
- Position the cursor to a specific field.

```
ATTR SCRA POS SS-NUM
```

- Change the intensity of a field.

```
ATTR SCRA BRT EMPL-NAME
```

- Apply multiple attributes to multiple fields.

```
ATTR SCRA PROT+BRT SS-NUM+EMPL-NAME
```

- If *fieldname* is part of a repeated block, it must be subscripted and the subscript included with the field name as passed to ATTR. The following example depicts SOC-SEC-NUM as part of a repeated block.

```
ATTR SCRA BRT+POS SOC-SEC-NUM(LINE-CTR)
```

Attributes, Screen Fields

Category: Screen Painter feature

Description: Assign field attributes by modifying the default attribute values for your text and I/O fields. To modify screen field attributes at run time, see *ATTR* and *CLEAR-ATTRS*.

Valid attribute values for screen fields on the Field Attribute screens are the following.

Attribute	Description and Values
Name	I/O field name; maximum 16 characters. Text fields do not have names because programs do not reference them. If you are retargeting an APS application to OS400, your field names can be only 10 characters long.

Hints:

- If you give a screen field the same name as its corresponding database field, APS Online Express automatically maps the relationship for you, prefixing the field name with the screen name; otherwise you must type the database field names in your program.

Attribute	Description and Values
	<ul style="list-style-type: none"> If the same field appears on several screens, give it the same name on each screen. APS lets you pass data between identically named fields on different screens during scenario prototyping and ISPF prototyping.
Length	Display field only; to change field length, move the cursor to the Xs designating the field and type in your changes. You can space over or delete the Xs representing the field, or extend the field with more Xs.
Intensity	B Bright N Normal D Dark
Type	U Unprotected (default); field is for both input and output. P Protected; field is output only.
MDT	Applies to IMS and CICS only. The modified data tag tells the terminal whether to return field data. When this tag is On for a field, the terminal always sends back data; when Off, the terminal returns data only if the user changes the data. On Default. Always send data, whether or not the end user modifies field; default for I/O fields. Off Send data only when end user modifies field; default for text fields. When you use field edits with an update program: <ul style="list-style-type: none"> For IMS, always set the tag On, Otherwise, results are unpredictable. For CICS, if you set the tag Off, you must set some variables in the CTRL file; otherwise, results are unpredictable. See <i>Control Files</i>.
Value	Initial value for screen field; maximum is field length or 27 characters, whichever is less.
APS edits	Display field indicating if any field edits were assigned to the screen field.

Attribute	Description and Values	
Num Lock	On	Activate keyboard numeric shift lock
	Off	Deactivate numeric shift lock (default)
	See also <i>Comments</i> .	
Light Pen	On	Light pen detectable.
	Off	Not light pen detectable (default).
Init cursor	No	Do not position cursor on this field when the program sends the screen. Default for all but the first I/O field.
	Yes	Position cursor on this field. Default for first I/O field.
	<p>If you change cursor positioning by setting a new field to Yes, you must change the previous "yes" field to No.</p> <p>By default Online Express positions the cursor on the function field for the non-repeated record block data. To override the default with the field you select here, blank out the Position Cursor on Field field with spaces on the Online Express Program Definition screen.</p>	
Color	B	Blue
	G	Green
	N	Neutral
	P	Pink
	R	Red
	T	Turquoise
	Y	Yellow
Highlight	B	Blinking
	U	Underline
	R	Reverse video
Modify	IMS only.	
	No	Program cannot modify extended attributes at run time (default).
	Yes	Program can modify extended attributes. APS generates the extra attribute bytes required.
	See also "Comments" below.	

Attribute	Description and Values
Format	For KANJI use only. Format field characters for a double-byte character set (DBCS) terminal. <i>blank</i> Single-byte characters only (default) D Double-byte characters only M Single- and double-byte characters combined
Ruledline	For KANJI use only. Place lines around the field on a DBCS terminal, as follows. <i>spaces</i> No lines B Surround field R Right side of field L Left side of field O Over field U Under field 00-0F See "Comments" below.

- Comments:**
- Under ISPF Prototyping, you cannot assign both the Protected and Dark attributes to I/O fields.
 - Turning the numeric keyboard locking attribute on does not ensure only numeric data is entered, because it is terminal dependent.
 - The Modify Extended Attributes attribute works in conjunction with the EXATTR MODIFBLE parameter on the Screen Generation Parameters: screen. If you set that parameter to F(alse), APS ignores the Modify attribute. If you set the parameter to T(rue), APS searches your screen to find which fields have the Modify attribute on.
 - To define a field to accept an MFS system literal, you give it the name of the literal, preceded by an asterisk (*). For example, for a field to contain the system literal DATE2, the field name should be *DATE2. When a field is used as a system literal, it is unavailable to the program and does not have modifiable attributes. See your MFS documentation for valid system literals.

- Using the values 00 through 0F for the KANJI ruled line attribute, surround the field as follows.

Value	Under	Right	Over	Left	
00					Equivalent to spaces
01	X				Equivalent to U
02		X			Equivalent to R
03	X	X			
04			X		Equivalent to O
05	X		X		
06		X	X		
07	X	X	X		
08				X	Equivalent to L
09	X			X	
0A		X		X	
0B	X	X		X	
0C			X	X	
0D	X		X	X	
0E		X	X	X	
0F	X	X	X	X	Equivalent to B

Bind and Translate Options, SQL

Compatibility SQL DB targets

Description: Define bind options for DB2 application and program generation.

Procedure 1 From the APS Options Menu enter option **5** in the **Command** field. Alternatively, from any APS screen enter **opt 5** in the **Command** or **Option** field. The APS Bind Options screen displays.

2 Select Bind and translate options as described below.

Field	Description and Values
DB2 System Name	Specify the appropriate name for your site. Default: DB2 .
Plan Name	Specify the plan name you use when you Bind an application. If you leave this field blank, the default depends upon your use of the BIND command in the Application Painter.
Owner of Plan (Authid)	Leave this field blank or specify a primary or secondary authorization ID of the BIND.
Qualifier	Leave this field blank or specify the implicit qualifier for the unqualified table names, views, indexes, and aliases contained in the plan.
Action	Specify the bind action to be executed. Valid values: add or replace.
Retain Execution Authority	Specify Yes if you specified REPLACE in the BIND ACTION field. Otherwise specify No.
Isolation Level	Valid values: rr or cs.
Plan Validation Time	Valid values: run or bind.
Explain Path Selection	Yes Activates the DB2 EXPLAIN function. No Does not activate the function.
Resource Acquisition Time	Valid values: use or allocate. If you enter ALLOCATE, you must enter DEALLOCATE in the Resource Release Time field.
Resource Release Time	Valid values: commit or deallocate. The value you enter in this field depends on the value you entered in the Resource Acquisition Time field.
Defer Prepare	Yes Generates the keyword DEFER(PREPARE), which defers the prepare statement referring to a remote object. No Default.

Field	Description and Values
Cache Size	Specify the size (in bytes) of the authorization cache to be acquired in the EDMPOOL for the plan. Valid values: 0 to 4096.
Data Currency	Yes Data currency is required for ambiguous cursors. No Data currency is not required for ambiguous cursors.
Current Server	Leave this field blank or specify a connection to a location before the plan runs.
Message Flag	Specify which messages display. Valid values: I, W, E, C, or blank.

Example:

```

COMMAND ==>
DATABASE (OS/2 DB MGR)    ==>          (If different from APPLICATION)
DB2 SYSTEM NAME          ==> DB2        (If different from APPLICATION)
PLAN NAME                 ==>          (If different from APPLICATION)
OWNER OF PLAN (AUTHID)   ==>          (Blank or one of the IDs)
QUALIFIER                 ==>
ACTION                    ==> REPLACE   (Add or Replace)
RETAIN EXECUTION AUTHORITY ==> YES      (Yes or No)
ISOLATION LEVEL           ==> RR        (RR or CS)
PLAN VALIDATION TIME      ==> BIND     (Run or Bind)
EXPLAIN PATH SELECTION    ==> NO       (Yes or No)
RESOURCE ACQUISITION TIME ==> USE      (Use or Allocate)
RESOURCE RELEASE TIME     ==> COMMIT   (Commit or Deallocate)
DEFER PREPARE             ==> NO       (Yes or No)
CACHE SIZE                ==>          (0 to 4096)
DATA CURRENCY             ==>          (Yes or No)
CURRENT SERVER            ==>
MESSAGE FLAG              ==>          (I, W, E, or C)

```

- Comments:**
- To reinstate the defaults defined for your site, enter **reset** in the **Option** field.
 - For detailed explanations of these fields, see the IBM DB2 Application Programming Guide.

CA

Category Program Painter and Specification Editor keyword (see *Keywords*)

Description: Redefine the TP-USERAREA field of the Commarea. To pass data between programs, see *TP-COMMAREA*.

Syntax: • Format 1:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
CA      datastructure
```

• Format 2:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
CA05   COBOLdatastructure
```

• Format 3:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
CADS   datastructurename
```

- Comments:**
- All formats generate an 05-level REDEFINES TP-USERAREA for the TP-USERAREA data structure and all other data elements at the same indentation level.
 - When using Formats 1 and 3, code the data structure in the Data Structure Painter format; when using Format 2, code in COBOL format.
 - A CA keyword in a batch program is not applicable and is ignored.
 - For IMS, ISPF Dialog, and CICS programs, Commarea data structures are generated in Working-Storage; for prototyping under ISPF, they are generated in Linkage.
 - &TP-USER-LEN controls the size of TP-USERAREA--for CICS, the default is 80 bytes; for IMS and ISPF Dialog, the default is zero. Change the size of this area by coding % &TP-USER-LEN=*nnn*, with a length greater than zero. Code this variable with the SYM1 or SYM2 keyword.

Example:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
IO      INPUT-FILE ASSIGN TO UT-S-INPUT
IO      OUTPUT-FILE ASSIGN TO UT-S-OUTPUT
```

```

FD      INPUT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 0 RECORDS
01      INPUT-REC          PIC X(80).
DS01    INPUTREC
FD      OUTPUT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 0 RECORDS
REC     OUTPUT-REC        X80
01      OUTPUT-REC-R REDEFINES OUTPUT-REC.
        ...COPY OUTREC.
CA      CA-INPUT-REC
        CA-IN-PART-NO      N8
        CA-IN-DESC        X50
        CA-IN-BASE-PRICE  N6V2
CADS    CAOUTREC

```

CCODE

Compatibility IMS DB target

Description: Use CCODE to include additional IMS command codes in DB-OBTAIN, DB-MODIFY, DB-PROCESS, and DB-STORE.

Syntax: Valid IMS CCODEs are:

P	Establish parentage at this level.
Q	Enqueue the segment.
U	Maintain current position at this level.
-	Null command code.

APS generates the following IMS CCODEs; do not code them yourself.

D	Put segment in I/O area (generated by REC).
F	Locate first occurrence (generated by FIRST).
L	Locate last occurrence (generated by LAST).
N	Do not replace segment (generated by REF).
V	Maintain current position at this level (generated by CURRENT).

For more information, refer to the applicable IMS manuals.

- Comments:**
- Most IMS command codes are generated by the call keywords; any command code you specify with the CCODE keyword is added to these.
 - Use CCODE carefully, because the command codes you specify with it are not validated by the APS/IMS DB Generator.

Example: Retrieve and enqueue RECORD-A and make it unavailable to other processing until the program terminates or explicitly frees the record with a checkpoint call.

```
DB-OBTAIN REC RECORD-A
... WHERE KEY-A = WS-KEY-A CCODE 'Q'
... REC RECORD-B
... WHERE KEY-B = WS-KEY-B
... VIEW PCBNAME RESET
```

Checkin

Category ENDEVOR Interface feature

Description: Add to or update the ENDEVOR library with an APS component from an APS Project Group. Alternatively, sign in a component at check in, without adding to or updating the ENDEVOR library. To retrieve a member from the library, see *Checkout*.

From the APS/ENDEVOR Version Control Menu, select option 1, Checkin. Alternatively, enter **CI** in the **Command** field on any APS screen.

Specify requirements on the Checkin screen, as follows.

Field	Description and Values
Entity Type	Entity type of the APS component to check in. Valid values are:
	ap Application Painter component in APSAPPL plus its related component in APRAPPL

Field	Description and Values
cn	Scenario Painter component in APSCNIO
ds	Data Structure Painter component in APSDATA
ox	Online Express component in APSEXPS
pg	Program Painter component in APSPROG plus its related component in APRPROG
rp	Report Mock-up Painter component in APSREPT
sc	Screen Painter component in APSSCRN
	For other APS component types in your Project.Group, specify a data set name, such as USERMACS and DDISYMB.
Member	Component name to check in, or leave the Member field blank to select from a member list.
System	ENDEVOR System name, if it differs from the default System name for your current APS Project.Group.
Subsystem	ENDEVOR Subsystem name, if it differs from the default Subsystem name for your current APS Project.Group.
Comment	Text comment for the check in.
CCID	ENDEVOR CCID for the check in.
Bypass Gen Processor	Specify yes to bypass the associated ENDEVOR Generate Processor.
Delete Input Source	Specify yes to delete the component from the APS Project.Group.
Processor Group	Name of the ENDEVOR Processor Group.
Override Signout	Specify yes to override an existing signout. You must have authority to do so.
Signin Only	Specify yes to Signin only, releasing a previous signout of the component issued with your user ID; the Add or Update action is not executed.
Stage	ENDEVOR Stage number for signin.

Field	Description and Values
Component Parts	<p>For checking in AP and PG component type components. Valid values are:</p> <p>none Default. Process only the component specified in the Member field.</p> <p>all Process the component specified in the Member field and all its associated component parts, or components.</p> <p>list Display the Component Types Selection screen, to select the associated component types for processing.</p> <p>APS submits a batch job to perform the check in when some or all component parts are checked in with the component specified in the Member field.</p>

Checkout

Category ENDEVOR Interface feature

Description: Retrieve and, by default, sign out a revision from a controlled member of the ENDEVOR library to an APS Project Group so that you can modify it. To add or update the library, see *Checkin*.

From the APS/ENDEVOR Version Control Menu, select option 2, Checkout. Alternatively, enter **CO** in the **Command** field on any APS screen.

Specify requirements on the Checkout screen, as follows.

Field	Description and Values
Entity Type	Component Type of the component to check out. Valid values same as for <i>Checkin</i> .
Member	Member name to check out, or leave the Member field blank to select from a member list.

Field	Description and Values
System	ENDEVOR System name, if it differs from the default System name for your current APS <i>project.group</i> .
Subsystem	ENDEVOR Subsystem name, if it differs from the default Subsystem name for your current APS Project.Group.
Stage	ENDEVOR Stage number of the member to check out.
Version	Default to the current revision. You can optionally override this value with another version number.
Level	Default to the current level. You can optionally override this value with another level number.
Comment	Text comment for the check out.
CCID	ENDEVOR CCID to associate with the check out.
No Signout	Specify yes to check out and browse the member without signing it out to your user ID.
Replace Member	Specify yes to overlay an existing member in the APS <i>project.group</i> .
Override Signout	Specify yes to override an existing Signout by another user. You must have authority to do so.
Component Parts	<p>For checking out AP and PG component type components. Valid values are:</p> <ul style="list-style-type: none"> none Default. Process only the component specified in the Member field. all Process the component specified in the Member field and all its associated component parts, or components. list Display the Component Types Selection screen, to select the associated component types for processing. <p>APS submits a batch job to perform the check in when some or all component parts are checked in with the component specified in the Member field.</p>

CIC-ADDRESS

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Use with the TP-LINKAGE call (*TP-LINKAGE*) to access CICS storage areas.

Syntax: `CIC-ADDRESS option(linkdataname) [option(linkdataname) ...]`

Parameters:

<i>option</i>	CWA	Common Work Area
	TCTUA	Terminal Control Table User Area
	TWA	Transaction Work Area
	CSA	Common Storage Area (z/OS target only)
	EIB	Execute Interface Block (z/OS target only)
<i>linkdataname</i>	01-level Linkage Section data area identical to the linkdataname in the associated TP-LINKAGE call.	

Example: Pass information to the application program. Set the Linkage Section data area (LK-CWA) in the current program to the address of the CWA for access.

```
-KYWD- 12-*----20---*-----30---*-----40---*-----50---*-----60
SYLK   TP-LINKAGE LK-CWA
LK01   LK-CWA                               PIC X(200).
NTRY   CIC-ADDRESS CWA(LK-CWA)
```

CIC-ASSIGN

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Obtain values outside the program and assign them to a Working-Storage data area in the current program.

Syntax: `CIC-ASSIGN CICOption(dataarea) [CICOption(dataarea)...]
... [ERROR(errorpara)]`

Parameters:

<i>option</i>	Valid CICS option. See your CICS reference manual for more information.
<i>(dataarea)</i>	COBOL data name containing the result of the call.
<i>(errorpara)</i>	User-defined error routine to perform when an abnormal condition occurs.

Example: Obtain a value outside the application program and assign it to a user-defined area.

```
CIC-ASSIGN APPLID(WS-APPLID)
... ERROR(ERROR-PARA)
```

CIC-CANCEL

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Cancel a previously issued CIC-DELAY (See *CIC-DELAY*) or CIC-START (See *CIC-START*).

Syntax: `CIC-CANCEL [REQID(name)]
... [TRANSID(name)]
... [SYSID(name)]
... [ERROR(errorpara)]`

Parameters:

<code>ERROR(errorpara)</code>	User-defined error routine to perform when an abnormal condition occurs.
<code>REQID(name)</code>	Unique call name; can be a literal or COBOL data name (maximum 8 characters).

<code>SYSID(name)</code>	Remote system name; can be a literal or a COBOL data name (maximum 4 characters).
<code>TRANSID(name)</code>	Transaction code identifying the program where control returns; can be a literal (maximum 4 characters) or COBOL data name (minimum 5 characters).

Example: Cancel an activity invoked by a CIC-START.

```
CIC-CANCEL TRANSID('TRAN') ERROR(ERROR-PARA)
```

CIC-DELAY

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Suspend task processing for a prescribed time interval.

Syntax: `CIC-DELAY [REQID(name)]`
`... [INTERVAL(hhmmss) | TIME(hhmmss)]`
`... [ERROR(errorpara)]`

Parameters:	<code>ERROR(errorpara)</code>	User-defined error routine to perform when an abnormal condition occurs.
	<code>INTERVAL(hhmmss)</code>	Time interval between issuing and executing a call. Hhmmss can be replaced by zero, a decimal constant, or a COBOL data name defined as PIC S9(07) COMP-3.
	<code>REQID(name)</code>	Unique call name; can be a literal or COBOL data name (maximum 8 characters).
	<code>TIME(hhmmss)</code>	Expiration time for the DELAY function. Hhmmss can be replaced by a decimal constant or a COBOL data name defined as PIC S9(07) COMP-3.

Examples: Suspend task processing for a 5-minute interval; let another task cancel this activity (UNIQCOM command).

```
CIC-DELAY INTERVAL(500) ERROR(ERROR-PARA) REQID('UNIQCOM')
```

Suspend task processing until 1:30 a.m.

```
CIC-DELAY TIME(013000) ERROR(ERROR-PARA) REQID('UNIQCOM')
```

CIC-DELETEQ-TD

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Delete all transient data associated with a predefined transient data queue.

Syntax: `CIC-DELETEQ-TD QUEUE(name)`
`... [SYSID(name)]`
`... [ERROR(errorpara)]`

Parameters:

<code>ERROR(<i>errorpara</i>)</code>	User-defined error routine to perform when an abnormal condition occurs.
<code>QUEUE(<i>name</i>)</code>	Queue name; can be a literal (maximum 8 characters), or COBOL data name (maximum 30 characters) defined as X(4).
<code>SYSID(<i>name</i>)</code>	Remote system name; can be a literal or a COBOL data name (maximum 4 characters).

Example: Delete all the intrapartition transient data stored in storage queue 'TDAQ'.

```
CIC-DELETEQ-TD QUEUE('TDAQ') ERROR(ERROR-PARA)
```

CIC-DELETEQ-TS

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Delete all temporary data and free all storage associated with a temporary storage queue.

Syntax: `CIC-DELETEQ-TS QUEUE(name)`
`... [SYSID(name)]`
`... [ERROR(errorpara)]`

Parameters:

<code>ERROR(<i>errorpara</i>)</code>	User-defined error routine to perform when an abnormal condition occurs.
<code>QUEUE(<i>name</i>)</code>	Queue name; can be a literal (maximum 8 characters), or COBOL data name (maximum 30 characters) defined as X(4).
<code>SYSID(<i>name</i>)</code>	Remote system name; can be a literal or a COBOL data name (maximum 4 characters).

Example: Delete all the data stored in temporary storage queue 'TSAQ'.

```
CIC-DELETEQ-TS QUEUE('TSAQ') ERROR(ERROR-PARA)
```

CIC-FREEMAIN

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Release storage previously acquired by a CIC-GETMAIN (See *CIC-GETMAIN*).

Syntax: `CIC-FREEMAIN DATA(linkdataname)`

Parameters: *linkdataname* 01-level Linkage Section data area identical to the *linkdataname* in the associated TP-LINKAGE call.

Example: Release the main storage of LK-STORAGE-AREA.

```
CIC-FREEMAIN DATA(LK-STORAGE-AREA)
```

CIC-GETMAIN

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Obtain and initialize main storage. To release storage, see *CIC-FREEMAIN*.

Syntax: `CIC-GETMAIN SET(linkdataname)`
`... LENGTH(value) | FLENGTH(value)`
`... [INITIMG(value)]`
`... [ERROR(errorpara)]`

Parameters: `ERROR(errorpara)` User-defined error routine to perform when an abnormal condition occurs.

`INITIMG(value)` Initialize storage area. Value is 1 byte; can be a literal or COBOL data name.

`FLENGTH(value)` Applies to the z/OS target only. Specify length as a full word value.

`LENGTH(value)` Maximum length of data; can be a literal (LINK or XCTL only) or COBOL data name defined as S9(04)COMP. Can also be a partial length (XCTL only).

`SET(linkdataname)` 01-level Linkage Section data area identical to the *linkdataname* in the associated TP-LINKAGE call.

Example: Obtain an area of main storage of the length specified in the Working-Storage data area (WS-LENGTH). Specify initialization value (WS-BLANK) for the acquired main storage.

```
CIC-GETMAIN SET( STORAGE-AREA)
... LENGTH(WS-LENGTH)
... INITIMG(WS-BLANK)
... ERROR(ERROR-PARA)
```

CIC-LOAD

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Load specified programs, tables, or maps from a library to main storage; to delete these, see *CIC-RELEASE*.

Syntax:

```
CIC-LOAD PROGRAM(name) [SET(linkdataname)]
... [LENGTH(dataarea)]|[FLENGTH(dataarea)]
... [ENTRY(pointref)]
... [HOLD] [ERROR(errorpara)]
```

Parameters:	ENTRY(<i>pointref</i>)	BBL cell containing program address after the LOAD operation.
	ERROR(<i>errorpara</i>)	User-defined error routine to perform when an abnormal condition occurs.
	FLENGTH(<i>value</i>)	Applies to the z/OS target only. Specify length as a full word value.
	HOLD	Hold loaded module in main storage until a CIC-RELEASE call executes.
	LENGTH(<i>value</i>)	Maximum length of data; can be a literal (LINK or XCTL only) or COBOL data name defined as S9(04)COMP. Can also be a partial length (XCTL only).

PROGRAM(<i>name</i>)	Name of module to load into main storage; can be a literal or COBOL data name (maximum 8 characters).
SET(<i>linkdataname</i>)	01-level Linkage Section data area identical to the linkdataname in the associated TP-LINKAGE call.

Example: Load table TAXTAB into main storage; perform user-defined error routine ERROR-PARA when an error occurs; store the address at which the module was loaded in Linkage Section data area TAX-TABLE-AREA; and store the length of the loaded module in Working-Storage data area TAX-LEN.

```
CIC-LOAD PROGRAM('TAXTAB')
... SET(TAX-TABLE-AREA) LENGTH(TAX-LEN)
... ERROR(ERROR-PARA)
```

CIC-READQ-TD

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Read transient data from a predefined data queue.

Syntax:

```
CIC-READQ-TD QUEUE(name)
... INTO(dataarea) [SET(linkdataname)
... [LENGTH(dataarea)] [SYSID(name)]
... [ERROR(errorpara)]
```

Parameters: ERROR(*errorpara*) User-defined error routine to perform when an abnormal condition occurs.

INTO(*dataarea*) Name of data area where APS places transient or temporary data.

LENGTH(*value*) Maximum length of data; can be a literal (LINK or XCTL only) or COBOL data name defined as S9(04)COMP. Can also be a partial length (XCTL only).

QUEUE(<i>name</i>)	Queue name; can be a literal (maximum 8 characters), or COBOL data name (maximum 30 characters) defined as X(4).
SET(<i>linkdataname</i>)	01-level Linkage Section data area identical to the <i>linkdataname</i> in the associated TP-LINKAGE call.
SYSID(<i>name</i>)	Applies to the z/OS target only. Remote system name; can be a literal or a COBOL data name (maximum 4 characters).

Example: Read a record from a transient data queue 'TDAQ' into data area WS-TD-REC.

```
CIC-READQ-TD QUEUE('TDAQ') INTO(WS-TD-REC)
... LENGTH(WS-TD-LEN)
... ERROR(ERROR-PARA)
```

CIC-READQ-TS

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Retrieve data from a temporary storage queue in main or auxiliary storage.

Syntax:

```
CIC-READQ-TS QUEUE(name)
... INTO(dataarea) | SET(linkdataname)
... LENGTH(dataarea) NUMITEMS(dataarea)
... [ ITEM(value) | next ]
... [ SYSID(name) ]
... [ ERROR(errorpara) ]
```

Parameters: ERROR(*errorpara*) User-defined error routine to perform when an abnormal condition occurs.

INTO(*dataarea*) Name of data area where APS places transient or temporary data.

ITEM(<i>value</i>)	Relative record number in the queue; can be a literal or COBOL data name defined as S9(04)COMP. Required with REWRITE.
LENGTH(<i>value</i>)	Maximum length of data; can be a literal (LINK or XCTL only) or COBOL data name defined as S9(04)COMP. Can also be a partial length (XCTL only).
NEXT	Read next sequential logical record.
NUMITEMS(<i>dataarea</i>)	Applies to the z/OS target only. Number of items in the queue.
QUEUE(<i>name</i>)	Queue name; can be a literal (maximum 8 characters), or COBOL data name (maximum 30 characters) defined as X(4).
SET(<i>linkdataname</i>)	01-level Linkage Section data area identical to the linkdataname in the associated TP-LINKAGE call.
SYSID(<i>name</i>)	Applies to the z/OS target only. Remote system name; can be a literal or a COBOL data name (maximum 4 characters).

Example: Read the next (or only) record from temporary storage queue 'TSAQ' into data area WS-TD-RECORD.

```
CIC-READQ-TS QUEUE (' TSAQ ' )
... INTO ( WS-TD-RECORD ) LENGTH ( WS-TD-RECLEN )
... ERROR ( ERROR-PARA )
```

CIC-RELEASE

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target in the z/OS environment

Description: Delete program, table, or map, previously loaded with a CIC-LOAD call (See *CIC-LOAD*), from main storage.

Syntax: CIC-RELEASE PROGRAM(*name*)
 ... [ERROR(*errorpara*)]

Parameters: ERROR(*errorpara*) User-defined error routine to perform when an abnormal condition occurs

PROGRAM(*name*) Main storage module to be deleted; can be a literal or COBOL data name (maximum 8 characters)

Example: Delete program PROG5.

```
CIC-RELEASE PROGRAM('PROG5')
... ERROR(ERROR-PARA)
```

CICS

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS targets

Description: Code and pass through native CICS calls.

Syntax: CICS *command*

Example: Retrieve the system time of day.

```
CICS ASKTIME
```

CIC-SCHEDULE-PSB

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS and IMS DB targets

Description: Schedule the PSB in the program, if not currently scheduled; to terminate it, see *CIC-TERM-PSB*. APS automatically generates these calls when you specify a PSB in the Application Painter.

Syntax: CIC-SCHEDULE-PSB

Comment: To define a PSB in your application definition, see the APS *User's Guide* chapter *Paint the Application Definition*."

CIC-SEND-TEXT

Category Data communication call (see *Data Communication Calls*)

Compatibility CICS target

Description: Format output data without mapping and transmit to a terminal or line printer.

Syntax: CIC-SEND-TEXT FROM(*dataarea*)
 ... LENGTH(*value*) [*CICOptions*]
 ... [ERROR(*errorpara*)]

Parameters:	<i>CICOptions</i>	Valid CICS option. See your CICS reference manual for more information.
	ERROR(<i>errorpara</i>)	User-defined error routine to perform when an abnormal condition occurs.
	FROM(<i>dataarea</i>)	Data area to be acted on.
	LENGTH(<i>value</i>)	Maximum length of data; can be a literal (LINK or XCTL only) or COBOL data name defined as S9(04)COMP; can also be a partial length (XCTL only).

Comment: Use CIC-SEND-TEXT prior to a TERM (See *TERM*) from a main program in order to clear the screen and unlock the keyboard.

Example: CIC-SEND-TEXT outputs data block TEXT-STRING, to be formatted without being mapped.

```
CIC-SEND-TEXT FROM(TEXT-STRING)
... LENGTH(100) ERASE FREEKB
... ERROR(ERROR-PARA)
```

CIC-SERVICE-RELOAD

Category Data communication call (see *Data Communication Calls*)

Compatibility CICS target

Description: Establish addressability to a data area in the Linkage Section following an address change in the BLL cell.

Syntax: `CIC-SERVICE-RELOAD linkdataname`

Parameters: *linkdataname* 01-level Linkage Section data area identical to the *linkdataname* in the associated TP-LINKAGE call.

Comment: SERVICE-RELOAD is an OS/VS COBOL statement; do not use with COBOL II, because the compiler treats this call as a CONTINUE statement.

Example: Establish addressability to Linkage area LINK-AREA-1 after an address change.

```
MOVE PASSED-ADDRESS TO LINK-AREA-1-PNTR
CIC-SERVICE-RELOAD LINK-AREA-1
```

CIC-START

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Start a task on a local or remote system at a specified time.

Syntax: `CIC-START TRANSID(name)`
`... [INTERVAL(hhmmss) | TIME(hhmmss)] [CICOptions]`
`... [ERROR(errorpara)]`

Parameters: <i>CICOptions</i>	Valid CICS option. See your CICS reference manual for more information.
ERROR(<i>errorpara</i>)	User-defined error routine to perform when an abnormal condition occurs.
INTERVAL(<i>hhmmss</i>)	Time interval between issuing and executing the call. Hhmmss can be replaced by zero, a decimal constant, or a COBOL data name defined as PIC S9(07) COMP-3.
TIME(<i>hhmmss</i>)	Expiration time for the START function. Hhmmss can be replaced by a decimal constant or a COBOL data name defined as PIC S9(07) COMP-3.
TRANSID(<i>name</i>)	Transaction code identifying the program where control returns; can be a literal (maximum 4 characters) or COBOL data name (minimum 5 characters).

Examples: Start a specific task (not associated with a terminal) in one hour.

```
CIC-START TRANSID('TRNL') INTERVAL(10000)
... ERROR(ERROR-PARA)
```

Initiate task TRN2 associated with terminal STA3; begin the task at 5:30 P.M.

```
CIC-START TRANSID('TRN2') TIME(173000)
... TERMID('STA3') ERROR(ERROR-PARA)
```

CIC-TERM-PSB

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Reference

Description: Terminate the currently scheduled PSB (See *CIC-SCHEDULE-PSB*). You normally use this call before transferring to another program.

Syntax: CIC-TERM-PSB

CIC-WRITEQ-TD

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Write transient data to a predefined data queue.

Syntax: CIC-WRITEQ-TD QUEUE(*name*) FROM(*dataarea*)
 ... [LENGTH(*value*)] [SYSID(*name*)]
 ... [ERROR(*errorpara*)]

Parameters: ERROR(<i>errorpara</i>)	User-defined error routine to perform when an abnormal condition occurs.
FROM(<i>dataarea</i>)	Data area to be acted on.
LENGTH(<i>value</i>)	Maximum length of data; can be a literal (LINK or XCTL only) or COBOL data name defined as S9(04)COMP. Can also be a partial length (XCTL only). Required with SYSID.
QUEUE(<i>name</i>)	Queue name; can be a literal (maximum 8 characters), or COBOL data name (maximum 30 characters) defined as X(4).
SYSID(<i>name</i>)	Remote system name; can be a literal or a COBOL data name (maximum 4 characters).

Example: Write data to predefined transient data queue 'TRDQ'.

```
CIC-WRITEQ-TD QUEUE('TRDQ') FROM(WS-MESSAGE)
... LENGTH(WS-TD-LEN)
... ERROR(ERROR-PARA)
```

CIC-WRITEQ-TS

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS target

Description: Write or rewrite temporary data records to a temporary storage queue.

Syntax:

```
CIC-WRITEQ-TS QUEUE(name) FROM(dataarea)
... LENGTH(value) [SYSID(name)]
... [ITEM(dataarea) [REWRITE] [CICOptions]]
... [NOSUSPEND]
... [ERROR(errorpara)]
```

Parameters: <i>CICOptions</i>	Valid CICS option. See your CICS reference manual for more information.
ERROR(<i>errorpara</i>)	User-defined error routine to perform when an abnormal condition occurs.
FROM(<i>dataarea</i>)	Data area to be acted on.
ITEM(<i>value</i>)	Relative record number in the queue; can be a literal or COBOL data name defined as S9(04)COMP. Required with REWRITE.
LENGTH(<i>value</i>)	Maximum length of data; can be a literal (LINK or XCTL only) or COBOL data name defined as S9(04)COMP. Can also be a partial length (XCTL only).
NOSUSPEND	Return to the program without waiting for resources to become available.
QUEUE(<i>name</i>)	Queue name; can be a literal (maximum 8 characters), or COBOL data name (maximum 30 characters) defined as X(4).
REWRITE	Overwrite existing record in queue with data contained in data area.
SYSID(<i>name</i>)	Remote system name; can be a literal or a COBOL data name (maximum 4 characters).

Example: Write a record to a temporary storage queue in auxiliary storage, where the queue name is in QUEUE-NAME.

```
CIC-WRITE-TS QUEUE(QUEUE-NAME)
... FROM(WS-TS-RECORD)
... LENGTH(WS-TS-LENGTH)
... ITEM(WS-TS-ITEM-NO)
... ERROR(ERROR-PARA)
```

CLEAR

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS, IMS DC, and ISPF Dialog targets

Description: Move spaces or low-values to all fields in a specified screen.

Syntax: [TP-|SC-]CLEAR *screenname*

Comment: This call does not alter the field attributes.

Example: Move spaces to all fields on screen SCRA.

```
CLEAR SCRA
```

CLEAR-ATTRS

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS, IMS DC, and ISPF Dialog targets

Description: Reset all screen field attributes to their original painted values.

Syntax: [TP-]CLEAR-ATTRS *screenname*

Comment: The screen field contents do not change.

Example: Reset all field attributes to their original values for screen SCRA.

```
CLEAR-ATTRS SCRA
```

COBOL/2 Support

Description: Code program logic in both the Program Painter and Specification Editor using COBOL/2 structures.

Under APS COBOL/2 support, you can:

- Code both S-COBOL and COBOL/2 structures in the same program.
- Code structures in columns 7 - 80 (rather than column 12 - 72).
- Import existing COBOL/2 program Procedure Division code directly into APS, without any changes.
- Generate COBOL/2 output, regardless of whether you use S-COBOL, COBOL/2, or a combination.
- Generate PERFORM THRU *paragraphname*-EXIT code for each paragraph. To do so, change the default NO setting for the GENEXIT flag to YES in the ISPSLIB member SSMCOMP.

You can turn COBOL/2 support on and off by setting the Generate COBOL/2 field on the Generation Options screen.

When programming in the Program Painter or Specification Editor, you can code in COBOL/2 as follows.

- To turn on COBOL/2, enter **cobol2** on the **Command** line.
- To turn off COBOL/2, enter **scobol** on the **Command** line.

- Comments:**
- APS saves the current status of COBOL/2 support; therefore it remains turned on or off the next time you access APS.
 - There are slight differences in the Program Painter and Specification Editor screens when COBOL/2 is turned on.

- All keywords are supported and required as before. Exception: The PARA keyword is optional; you can code the paragraph name beginning in column 8. Keywords are coded in columns 8 - 11 (area B).
- Paragraph names can have a maximum of 24 characters.
- Code only one verb per line.
- Nested COBOL/2 programs are not supported.
- You must use consistent indentation (we recommend four spaces) throughout your program, even when importing COBOL/2 structures, or you will receive warning messages and your generated logic will probably be inaccurate. For example:

```
000001  IF OK-TO-PROCEED
000002      MOVE A TO B
000003      IF D = E
000004          MOVE F TO G
000005      ELSE
000006          MOVE H TO I
```

When compiling and generating the above code, APS will not know which IF statement lines 005 and 006 belong to.

- If a COBOL/2 EVALUATE statement is also valid S-COBOL EVALUATE syntax, APS processes it as an S-COBOL statement.
- When importing existing COBOL/2 code into APS, remember to assign the appropriate APS keywords to non-Procedure Division statements.
- APS stores COBOL/2 program files in a slightly different format to support access up to column 80. When an existing S-COBOL program is subsequently saved under COBOL/2, the source text of keyword lines shifts one byte to the right. This truncates any character coded in column 72 and displays a Truncation has occurred warning.

This truncation only occurs on a keyword line, and in fairly unusual circumstances, such as a PARA keyword followed by a long string of arguments with the last argument ending in column 72. If it happens, edit the member, fix the line, and save the file with COBOL/2 support turned on.

CODE

Category: Report Writer clause (see *Report Writer Structures* and the APS *User's Guide* chapter *Create Reports with Report Writer*)

Compatibility: Batch environments

Description: Specify a two-character literal placed at the beginning of each report line. It is useful when writing multiple reports to one file.

Syntax: `CODE literal`

Parameters: *literal* A two-character, non-numeric name inserted as the first two bytes of each report record or print line

- Comments:**
- The two bytes identifying a literal are included in the logical record size, not in the print line description.
 - If your report has a File Description (FD) in the REPORT IS clause, include a RECORD CONTAINS clause that allows for the two extra bytes the CODE clause needs; the default value of 133 for RECORD CONTAINS does not.

If your report does not have a File Description, the APS default record length of 250 includes the extra two bytes the CODE clause needs.
 - If you specify CODE for one report, specify it for all reports in the file.

Example: See the APS *User's Guide* chapter *Create Reports with Report Writer*.

Comments

Description: Enter comments in your program. To write comments in the Identification Division, see *REM*.

Syntax:

- Format 1, code anywhere in your program:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
/*      commentline
```

- Format 2, code in the Procedure Division only:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
/*commentline
```

- Format 3, code in the Procedure Division only:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
Program code /*comment
```

- Format 4, code in Customization Facility macros:

```
/* comment
```

- Format 5, code in the Data Structure Painter only:

```
* comment
/* comment
/* comment
```

Comments:

- Comment lines do not affect the function of a program or data structure. In the Program Painter, the last keyword entered before your comment line(s) is still in effect.

Note: You cannot code comments within database calls, data communication calls, or Data Structure Painter constructs.

- Do not use evaluation brackets in Formats 2, 3, and 4.
- Begin each comment line with */** or */**.
- In the Procedure Division, you can code */** anywhere in the column 12 - 72 area.

- A Customization Facility comment (Format 4) can start in any column, as long as it is on a line by itself and is indented (we recommend four spaces) underneath the Customization Facility statement.
- To code macro statements, use Format 4.
- Do not code comments within database calls.

Examples: In the Program Painter:

```
-LINE-  -KYWD-  12--*--20---*-----30-----*---40---*-----50---*-----
60
002000          /* S-COBOL COMMENT LINE
002010 /*      COBOL COMMENT LINE
002020  PARA    MAIN-PARA    /* S-COBOL COMMENT
```

In the Data Structure Painter:

```
-LINE-  -----  DATA STRUCTURE PAINTER  -----
000001          /* WORK FIELD 1 RECORD
000002          WRK1-FIELD-1    X(5)
000003              88 OPEN-VAL    V'OPEN'
000004              88 CLOSED-VAL  V'CLOSE'
```

Generated COBOL code:

```
000001  /* WORK FIELD 1 RECORD
000002  01  WRK1-FIELD-1    PIC X(5).
000003      88  OPEN-VAL    VALUE 'OPEN'.
000004      88  CLOSED-VAL  VALUE 'CLOSE'.
```

Component List (MS01)

Category: APS-generated report (see *Application Reports*)

Description: The Component List Report catalogs all of the components created for an application within a specific Project and Group.

The report lists components by painter in six columns - one each for applications, programs, screens, report mock-ups, data structures, and

scenarios. Within each column, the report lists components in alphabetical order. The bottom of the report totals the number of components listed for each painter. You can produce a report listing application components of one or more types.

- Comments:**
- Produce the Component List Report from the Documentation Facility.
 - This report does not provide information about subschemas or about the contents of each listed component.

Example:

```
REPORT CODE: MS01                APS APPLICATION DICTIONARY
PAGE          1
                                COMPONENT LIST
05/17/92 09:13                  MKTAPS.MKT2
SELECTION CRITERIA: ALL
*****
*****
```

APPLICATIONS	PROGRAMS	SCREENS	REPORTS	DATA STRUCTURES	SCENARIOS
CICSJSS	ADEMO	ADEMO	\$APSCMR	APCOMM	APDEMO
CICSVSAM	AW02PGM	ADEMOKB	MANUFAC	AWO3	APSDEMO
CUSTORDR	AW03PGM	ADEMO1	MWPART	BANK2	APSDEMO2
DEMOKEB	XXXPGM	ADEMO1J	REPORT1	COMAREA2	APSDEMO3
DEMO1803	ANNER1	ADEMO1X	VNDROPD	DLGAPPL	DEMO4
DLXVAPPL	DLGINQ	ADEMO3D		CPFDTAB	CBIS
DLX2APPL	LGMNU	ADEMO3J		DB2DEMO	DL2APPL
DMVAPPL	DLXVINQ	A2CASE		MARIA	DEMO###
LEVEL30	DLX2UPD	CMKTEMP		QSSGLOBL	ISR00007
MVS20SCR	DMOMNU	CMK1		QSS9	MVS21
PTSUNLD	PMINFOT1	DLGU			SSS2
PXAPPL	PMINFOT2	DMOM			TEST
QSSAPPL	PMINFO3	KEB			
SSKTEST	PMINFO4	KEBDEMO1			
TDDEMO	PMUDS1	KEB1			
TESTQSS	PMXXXXT2	MEAD1			
USRDEMO	PMXXXXT3	P22604			
	PM1SVKEY	REWDEMO			
	PTSUNLD	TDCS			
		PXCUSTM	TDDDH		
		PXMENU	TDDST		
		PXORDRM	TDFIRN		

PXORDRS	TDJH
PXPARTL	TDME
PXVCUSTM	TDOJ
PXVMENU	TDOM
PXVORDRM	TDOT
PXVORDRS	TDOU
PXVPARTL	TDPF

.
.

TDCM
TDCS
TDME
TDOJ
TDOM
TDOT
TDOU
TDPF
TDPL
TDPM

APPLICATIONS	-	28
PROGRAMS	-	60
SCREENS	-	50
REPORTS	-	6
DATA STRUCTURES	-	13
SCENARIOS	-	18

CONTROL

Category: Report Writer clause (see *Report Writer Structures* and the *APS User's Guide* chapter *Create Reports with Report Writer*.)

Compatibility: Batch environments

Description: Identify control data items (controls), which are tested for a change each time a detail line is printed. Create a control hierarchy for the report control headings and footings (control breaks).

Syntax: • Format 1:

```
CONTROL [IS] [FINAL] dataname
```

- Format 2:

```
CONTROLS [ARE] [FINAL] dataname1 ... datanameN
```

- Parameters:**
- | | |
|-----------------|--|
| <i>dataname</i> | Data item that causes a control break when it changes. |
| FINAL | Inclusive report control group not associated with a control <i>dataname</i> . It represents the highest level of control. |
- Comments:**
- *Datanames* can be qualified, but cannot be subscripted or indexed, or have a subordinate data item with a variable size defined in an OCCURS clause. Each *dataname* must be a different data item.
 - List the *datanames* from the highest to lowest level. An implicit FINAL is the highest control. The first *dataname* is the major control; the last *dataname* is the minor (lowest) control.
 - You can omit the CONTROL clause when the only control is FINAL.
 - The first time a GENERATE builds a report, all control values are saved. The next GENERATE tests controls for changes every time a detail line is printed, by comparing the contents with the contents saved from the last GENERATE of the same report, as follows.
 - If the control is numeric, the relation test compares two numeric operands.
 - If the control is an index, the test compares two index data items.
 - If it is neither numeric nor an index, it compares two non-numeric operands.
 - A change in *dataname* (a control) causes a control break that:
 - Prints control footings for lower level *datanames*, followed by the control footing for the *dataname* causing the control break
 - Clears counters associated with the *datanames*
 - Prints control headings for the *dataname* and lower level *datanames*
 - Prints the detail line causing the break

Example: See the APS *User's Guide* chapter *Create Reports with Report Writer*

Control Files

Description: Use APS control file variables to control certain functions. Each control file contains documentation on its variables. Look in the APS CNTL PDS or library for these files.

APS CNTL File	Environment or Function Controlled
APCICSIN	CICS
APSDBDC	Database and data communication calls
APFEIN	Field edits
APDLGIN	ISPF Dialog
APIMSIN	IMS DB and DC
APDB2IN	SQL
APVSAMIN	VSAM
APHLPIN	User Help Facility database

CICS

File APCICSIN controls:

- DLIUIB name
- Prologue generation
- Inline error checking (for example, CICS IGNORE globally or NOHANDLE on a call-by-call basis)
- Data name suffixes generated for field length and attribute data names
- DFHAID and APS-EIBRCODE copybook inclusion
- EJECTs
- CICS comments
- APS/CICS call overrides
- RESP/RESP2 options

DB and DC Calls

Note: Changing APSDBDC file parameter values affects the entire installation. To override values only for a specific Project, set the variables in the CNTL file APSPROJ.

Parameter	Description
% SET NOBLANK	Suppress blank lines from appearing in the output. Override with % SET BLANK.
% SET EVAL-BRACKETS "<>"	Define the characters used as evaluation brackets. Important: Overriding this parameter is not recommended because it will affect the Customization Facility macros.
% SET LOOP-LIMIT 500	Limit the APSMACS (APS macros) and USERMACS (user macros) Customization Facility loop structures to a maximum of 500 loops. Override with another number, 6 digits maximum. <hr/> Note: Loop limit flags for DB-PROCESS loops are in the target-specific APS CNTL files. <hr/>
% &MACRO-COMMENTS = 1 and % &TP-MACRO-COMMENTS = 1	Allows COBOL comments to appear in generated source. Override with 0.
% SET TRACE ERROR	Customization Facility parameter. An error trace mechanism that identifies the line of source that caused the error, the active % INCLUDE statement(s), the macro(s) currently invoked and not yet ended, and the number of loops completed at the time of error (if applicable). The severity codes of errors traced are F (Fatal), E (Error), W (Warning), and I (Information) messages. To eliminate Information message traces, append a space and the keyword NOINFO to the trace statement. To turn off the trace in selected portions of a program, code % SET NOTRACE.

Parameter

% &IM-HOLD-DEFAULT = "NOHOLD"

Description

Prevent APS-generated IMS DB-OBTAIN calls from HOLDing a record for update except when HOLD is specified in the DB-OBTAIN call. Override with "procopt", which does HOLD a record automatically, assuming the subschema or PSB specifies that the program can update the record.

% &VS-PROTOTYPE-MODE = "NO"

"Yes" specifies that a VSAM program is a prototype, enabling you to code DB calls without accessing a VSAM file. All subschema validation at program generation is still performed. NO means prototype mode is inactive.

% &IM-SUPPRESS-DB-CALL = "NO"

Deactivate the prototype mode for an IMS program. "Yes" specifies that an IMS program is a prototype, enabling you to code DB calls without accessing a data base. All subschema validation at program generation is still performed.

% &IM-USE-DFS0AER = 0

Disable the APS-supplied IMS database error routine macros from calling the IMS-supplied error display module, DFS0AER. All database errors are resolved using only the APS-supplied status flags. To enable use of DFS0AER and the APS-supplied flags, set to 1. If your installation does not use DFS0AER, leave this flag set to 0. See *Error Handling*.

% &GEN-DB-REC-01-NAMES = 0

IMS DB parameter. If your top-level copylib records begin with the level number 01 positioned in column 8, use the flag default of 0. If they don't, override with 1. When writing DDI statements for copylibs that don't begin with 01-level records, see instructions in the topic *Writing DDI Statements for IMS* in the *APS User's Guide*.

Note: For VSAM, an equivalent flag exists in the APS CNTL file APVSAMIN.

Parameter	Description
% &IM-SUPPRESS-COPYLIB = 0 and % &VS-SUPPRESS-COPYLIB = 0	Override with 1 to suppress a copylib described in your DDI statements. &IM is for IMS; &VS is for VSAM.
% &IM-USE-ASSEMBLER-BLOCKING = 1	Use a supplied Assembler routine to move data between record I/O areas and IMS concatenated segment I/O areas in APS-generated programs. This method is recommended because it's more efficient and doesn't require redefining the record I/O areas.
	<hr/> Note: Because this module is linked during generation of COBOL code, setting this variable to 1 might interfere with transportability of the generated code. <hr/>
	Overriding with 0 causes APS-generated programs to block and unblock IMS concatenated I/O areas using a byte-level loop in the generated code. Setting to 0 also causes APS to automatically redefine all record I/O areas as an array of bytes Note: Setting to 0 is incompatible with copylib members containing a COBOL OCCURS DEPENDING ON clause or multiple record declarations in a single member.
% ©-SECTION = "WORKING-STORAGE"	Specifies that copylib members are placed in Working-Storage.
% &TP-RETRY = 1	Set the parameter RETRY as the default parameter for the NTRY call. Override with 0 to set the default to NORETRY. You can also override on a program-by-program basis by coding RETRY or NORETRY in the NTRY call. See <i>NTRY</i> for more information.
% &IM-EXEC-DLI = 0	Generate CALL 'CBLTDLI' syntax. Override with 1 to generate EXEC DLI syntax.
% &DBCS = 0	Set to 1, the Double Byte Character Set flag, for KANJI support.

Field Edits

File APFEIN controls:

- Setting the maximum number of input and output edited fields per paragraph. The default is ten input fields and ten output fields. To change the defaults, set the &FE-INP-PARA-BREAK and &FE-OUT-PARA-BREAK variables as desired.
- Preventing execution of a user-defined input application edit when the field does not pass assigned field edits. The default is NO, which continues execution. To prevent execution, set the variable &FE-BYPASS-INPUT-APPL-EDIT to YES.
- Preventing execution of subsequent input and output field edits if a field fails a user-defined input application edit. The default is NO, which continues execution. To prevent execution, set the variables &FE-BYPASS-EDITS-IF-APPL-FAIL and &FE-BYPASS-OUTPUT-EDITS to YES. Overriding this parameter is not recommended because it affects Customizer rules.
- Using the USA format or European format that reverses the comma and decimal point. The default is the USA format. To use the European format, set the &FE-DECIMAL-IS-COMMA variable to YES.
- Supporting DBCS (Double Byte Character Set) characters. The default for the &FE-DBCS-ENABLE variable is YES.
- Resetting attribute bytes to the painted values if the field passes all assigned input edits under CICS. The default for the &FE-RESET-CICS-ATTRS variable is YES.
- Resetting field attribute bytes before or after input edits under CICS. The default for the &FE-RESET-CICS-ATTRS-AT-TOP variable is NO, which means after input edits.
- Allowing spaces within numbers. The default for the \$FE-EMB-SPACE-IN-NUMERIC is NO.
- Performing numeric calculations where the numeric de-edit option was not selected. The default is NO. Setting the &FE-EDIT-WHEN-NUMERIC to YES readies all numeric fields on the screen for calculations.
- Interpreting the century for output date formats. The default for the &FE-FIRST-YEAR-OF-CENTURY is 10. This means that APS interprets any date with a YY (year) value of 11 through 99 as being in the 20th century, that is, year 1911 through 1999, and any YY

value of 00 through 10 as being in the 21st century, that is, year 2000 through 2010.

IMS DB and DC

Use the IMS control file, APIMSIN, to control these functions:

- Prologue generation
- EJECTs
- Color, Blink, Reverse Video, Underscore switches
- System Service Calls (IM-STAT, IM-LOG, IM-ROLL, IM-ROLB)
- GSAM RSA calls
- DB-PROCESS call loop limit flag, &VS-IMS-LOOP-MAX (default 100); you can change the limit number or disable the check for the limit.

ISPF Dialog

Use the ISPF Dialog control file, APDLGIN, to control these functions:

- DYNAM or NODYNAM calls
- VDELETE variables upon TP-TERM
- TP-LINK and TP-XCTL generation of COBOL CALLs or ISPEXEC SELECTs
- Automatic open/close for VSAM files (enable/disable)
- DB2 COMMIT generation for ISPF Dialog TP-LINK, TP-XCTL, and TP-TERM calls
- Prologue generation
- EJECTs
- ISPF comments

SQL

Use the DB2 control file, APDB2IN, to control these functions:

- Comment generation (default: comments are generated)
- EJECTs

- Size of SQL calls accepted (default: 200 tokens, for example, items separated by spaces)
- Generation of the IBM DCLGEN convention, *DCLtablename*, for host 01-level qualifiers
- Automatic error processing, controlled by the variable &D2-AUTO-ERROR-HANDLING (default: ON)
- Customization of automatic error processing, including:
 - Modifying the APS error processing paragraph &D2-ERROR-PARA, which resides in the macro \$DB2-ERROR-SETUP
 - Modifying which status codes should be considered error conditions, in the macro \$DB2-CHECK-RETURNS-AUX, and changing the status of a referential integrity constraint from an abnormal condition to an invalid key
- The DB-PROCESS call loop limit flag, &DB2-LOOP-LIMIT (default: 100); you can change the limit number or disable the check for the limit.
- COMMIT generation for the DB-MODIFY, DB-STORE, DB-ERASE, and DB-CLOSE calls
- DB2 customization exits, which are program locations where you can write your own macros to customize APS/SQL calls
- Generated data names. You can:
 - Override the use of the *IND-cursorname* structure when generating indicator variables
 - Use an 01-level name instead of *copylib-REC* for DB2 record host-variable qualification
 - Modify generated data names used by \$DB2-EPILOGUE, which is invoked after all DB2 calls have been processed
 - Use a table alias name instead of the table name
- Comment generation for the APS \$DB2- calls, which are used by the APS/SQL calls

VSAM Batch and Online

Use the VSAM control file, APVSAMIN, to control these functions:

- Functions for both CICS/VSAM and batch:
 - Abnormal error processing (enable/disable automatic APS routine)
 - Non-referenced record descriptions (include/exclude)
 - Record inclusion
 - A flag you must use if your copylib records don't begin with the 01-level number
- CICS/VSAM-specific functions:
 - Abnormal error processing (exclude certain CICS Exceptional Conditions and ISI-Errors from being abnormal conditions)
 - DB-PROCESS call loop limit flag, &VS-CICS-LOOP-MAX (default 100); you can change the limit number or disable the check for the limit.
 - ENDBR/UNLOCK
 - Customization exits, which are program locations where you can write your own macros to customize APS/VSAM and native VSAM calls
 - Data name generation
 - Comment generation
- Batch-specific functions:
 - Abnormal error processing (exclude certain batch conditions from being abnormal conditions)
 - DB-PROCESS call loop limit flag, &VS-MVS-LOOP-MAX (default: 999,999); you can change the limit number or disable the check for the limit
 - Termination method (STOP RUN or call your own abend program)
 - Customization exits, which are program locations where you can write your own macros to customize APS/VSAM calls

- Data name generation
- Comment generation

User Help

Use the APS User Help Facility control file, APHLPIN, to control these functions for generating the User Help database:

- Program and screen names (if naming conflicts exists)
- Internal and external storage database targets
- Subschema access used by the help database
- Database name and attributes
- Database field names--COBOL or native
- Screen data storage options
- Data field length
- Global screen message field name
- Field help indicator string
- Date format
- PF key designations
- COBOL help invocation conditions
- APS-generated User Help comment suppression

Control Points

Category: Online Express feature

Description: Write and execute custom processing logic to supplement or override the default logic that Online Express generates. Execute this logic at any of the APS-provided locations in your program, known as program control points.

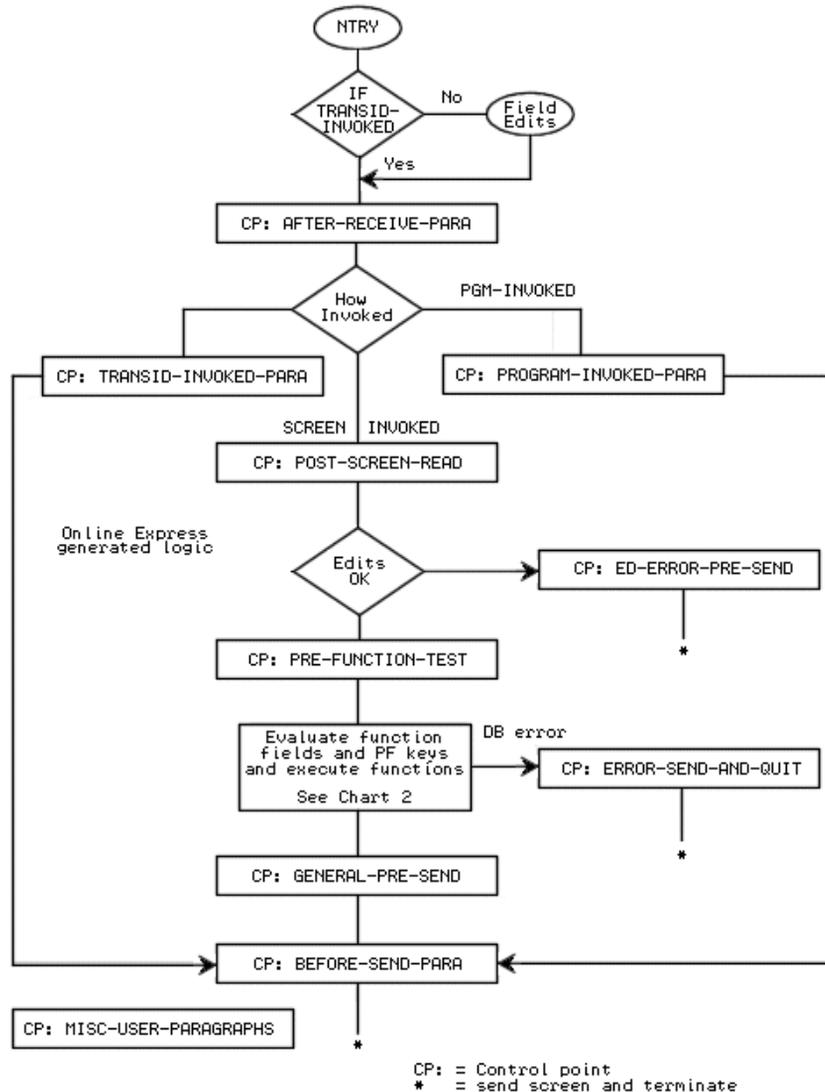
To view the control points in your program, display the Control Points screen or the Database Call Tailoring screen. Or, you can look in your generated program source to see where the control points occur. The complete set of control points is as follows.

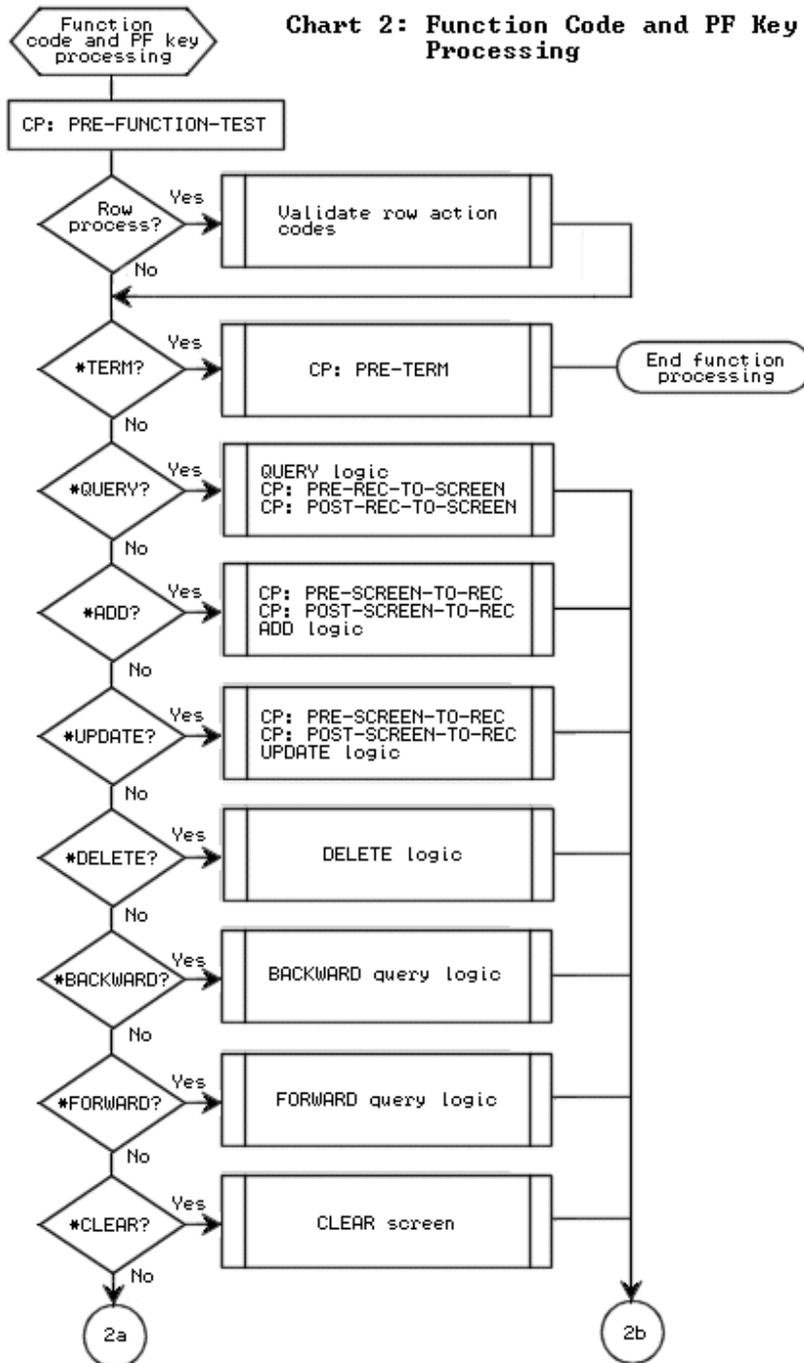
Control Point	Location in Program
After-Receive-Para	After entering a program, regardless of invocation mode.
Post-Screen-Read	After a screen-invoked program receives its screen.
Transid-Invoked-Para	After a transid-invoked program is invoked.
Program-Invoked-Para	When APS displays the screen of a program invoked by the XCTL or MSG-SW function.
Pre-Term	Before APS terminates the program.
After-Enter-Check	After the end user presses the processing key (the Enter key is the default), and before the PRE-FUNCTION-TEST paragraph executes.
Pre-Function-Test	Before APS evaluates all functions except the Terminate, or Exit, function.
Pre-Branch	Before each MSG-SW, XCTL, or Call function executes.
Ed-Error-Pre-Send	Before APS send a screen whose field edits have failed.
General-Pre-Send	After APS checks all functions, and before the TP-SEND call executes, when invocation mode is screen-invoked.
Before-Send-Para	Before APS sends the screen, regardless of invocation mode.
Pre-Screen-To-Rec	Before APS performs the MOVE-SCREEN-TO-REC paragraph.
Post-Screen-To-Rec	After APS performs the MOVE-SCREEN-TO-REC paragraph, and the Update or Add function executes.
Pre-Rec-To-Screen	Before APS performs the MOVE-REC-TO-SCREEN paragraph.
Post-Rec-To-Screen	After APS performs the MOVE-REC-TO-SCREEN paragraph, and after the Query function executes.

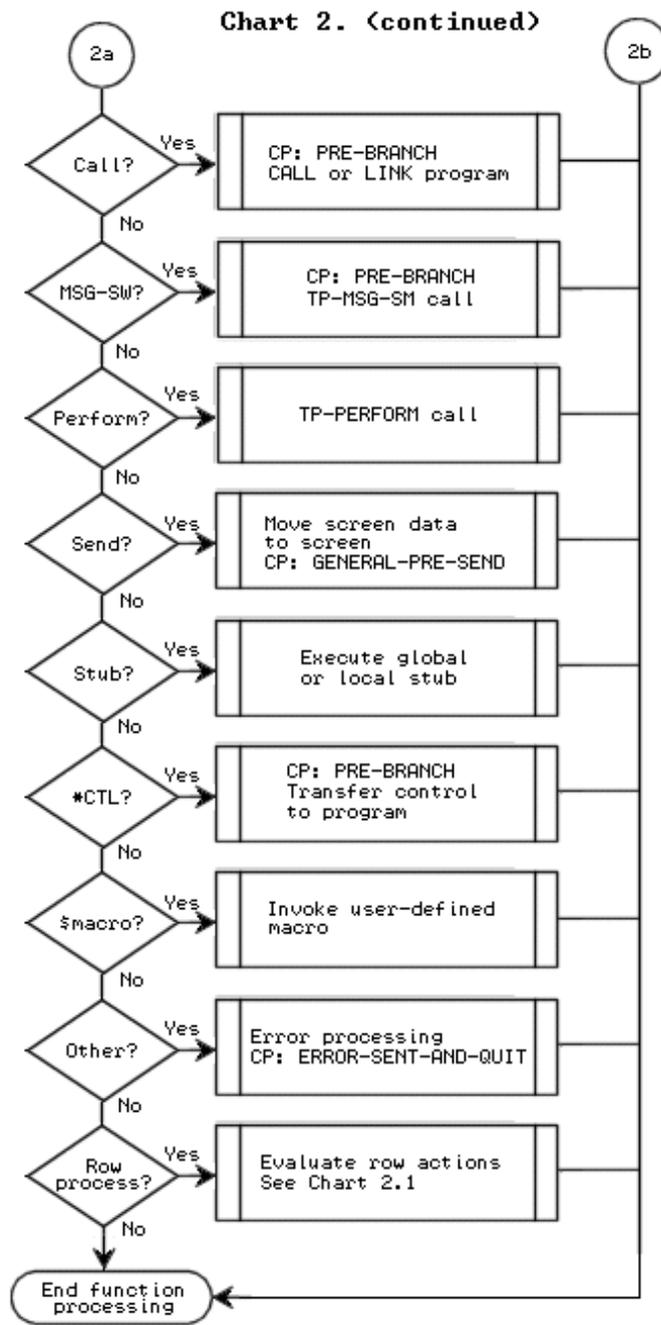
Control Point	Location in Program
Pre-RB1-Row-To-Rec	Before the Add or Update function executes for a repeated record block row, and before screen fields move to database fields. APS uses the subscript CTR to reference repeated block rows.
Post-RB1-Row-To-Rec	Before the Add or Update function executes for a repeated record block row, and after screen fields move to database fields. APS uses the subscript CTR to reference repeated block rows.
Pre-Rec-To-RB1-Row	After the Query or Forward function executes for a repeated record block row, and before database fields move to screen fields. APS uses the subscript CTR to reference repeated block rows.
Post-Rec-To-RB1-Row	After the Query or Forward function executes for a repeated record block row, and after database fields move to screen fields. APS uses the subscript CTR to reference repeated block rows.
Error-Send-And-Quit	When a program terminates abnormally, such as when a database call fails when the Database Call Tailoring screen's Abort On Error parameter is set to Y.
Misc-User-Paragraphs	A location where you can write and store any number of paragraphs to perform at any control point in the program. Code all your paragraphs in one file in this location.
Before DB Access	Before a non-loop database call executes
Before Loop	Before a loop database call executes
Normal Status (Before Record is Processed)	Before Online Express maps looped records to the screen
Normal Status	After Online Express maps any records to the screen
Exception Status	After the database call returns a status flag with the Exception status code
Error Status	After the database call returns a status flag with the Error status code

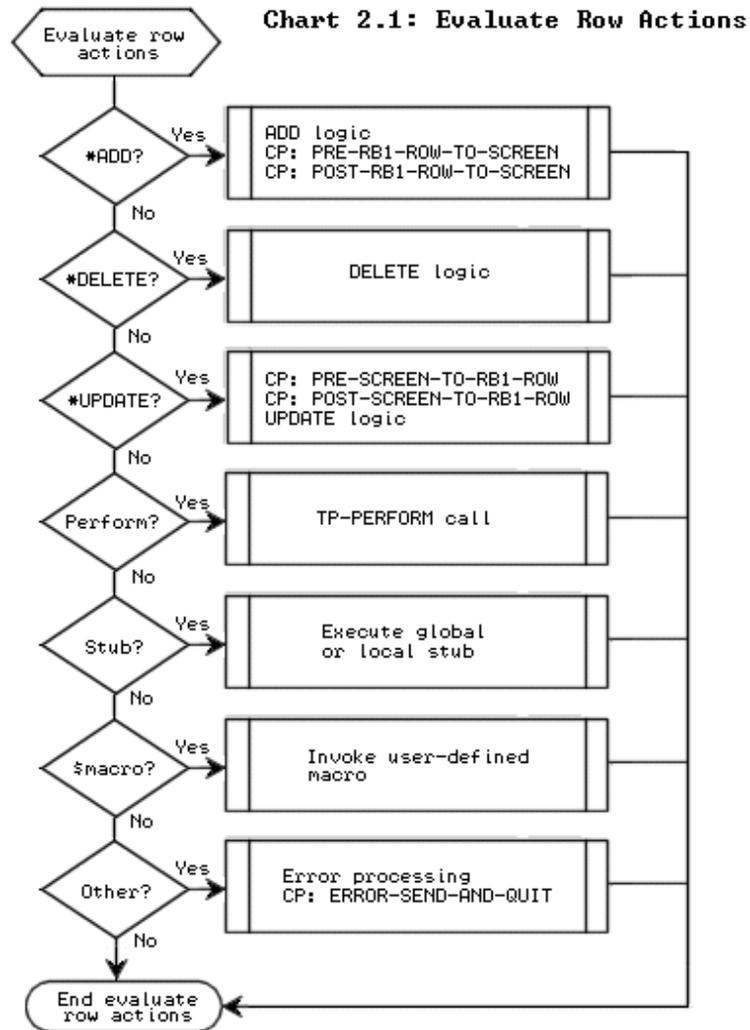
Control Point	Location in Program
After DB Access	After a non-loop database call executes
After Loop	After a loop database call executes
Flowcharts	The following flowcharts illustrate the locations of all control points in APS-generated programs.

Chart 1: Standard Control Points









The flowchart below illustrates the location of the database call control points in generated programs.

Non-Loop Call

```

CP: BEFORE DB ACCESS

IF OK-TO-PROCEED
  database call
  IF OK-ON-REC
    CP: NORMAL STATUS
  ELSE-IF exception-status
    CP: EXCEPTION STATUS
  ELSE
    CP: ERROR STATUS

MOVE MESSAGE TO SCREEN
PERFORM ERROR SEND AND QUIT

CP: AFTER DB ACCESS

```

Loop Call

```

CP: BEFORE LOOP

IF OK-TO-PROCEED
  DB-PROCESS call
  CP: NORMAL STATUS
  (BEFORE REC MAPPED TO SCREEN)

  IF OK-TO-PROCEED
    ADD 1 to CTR
    CHECK CTR
    PERFORM STOREKEY
    CP: NORMAL STATUS

  PERFORM MOVE REC TO SCRBLKS

CP: AFTER LOOP

```

- Comments:**
- Because the design and functionality of many programs differ, you often see a different subset of control points from program to program.
 - APS generates comments in your program to identify the control points. To activate or deactivate the generation of these comments in Online Express, access the Express Parms screen and enter yes in the Control Points Comments field.

- The Normal Status (Before Record is Processed) control point lets you add custom logic before looped records map to the screen. Use this control point to map only some of the records that a loop obtains. In your stub or macro, write conditional logic to determine which records to map. APS provides a flag, OK-TO-PROCEED, that you set to True to map and process the record, or False to bypass mapping and processing. You can ignore the flag if you do not use this control point; the flag is set to True by default. To add custom logic after APS maps any record to your screen, use the Normal Status control point.

Example: Map records that show annual sales of \$100,000 or more in the Northwest region, and calculate and map the grand total of those records. First define a loop call and qualify it to obtain the records of \$100,000 or more. Then tailor the loop call with two local stubs. The first stub checks the records obtained by the loop to allow only records of the Northwest region to be processed further. The second stub calculates the grand total of those records, and maps the total to the screen.

```
DB-PROCESS REC SALES-RECORD
... WHERE ANNUAL-SALES-TOTAL > 99999
    PERFORM CHECK-BEFORE-MAPPING-STUB-PARA
    IF OK-TO-PROCEED
        ADD 1 TO CTR
        PERFORM RECORD-STOREKEY-PARA
        MOVE REC-TO-SCREEN-BLK1
        PERFORM CHECK-AFTER-MAPPING-STUB-PARA
    .
    .
    .
CHECK-BEFORE-MAPPING-PARA
    TRUE OK-TO-PROCEED
    IF SALES-REGION NOT = NORTHWEST
        FALSE OK-TO-PROCEED
CHECK-AFTER-MAPPING-PARA
    calculation and mapping routine for grand total
```

Note:

- The CHECK-BEFORE-MAPPING-PARA paragraph executes at the Normal Status (Before Record is Processed) control point.
 - The CHECK-AFTER-MAPPING-PARA paragraph executes at the Normal Status control point.
-

Database Calls

Compatibility: All targets

Description: APS Logical View Database (DB) calls are predefined, easy-to-use statements with common syntax that allows transparent access to a variety of databases. The APS Logical View DB calls let you focus on what needs to be accomplished, rather than the mechanics of the target environment. To address environment-specific requirements, you can extend these calls with keywords; no native coding is required. These calls facilitate both generic processing and specialized requests.

APS supports the following database targets:

- IDMS
- IMS DB
- SQL
- VSAM batch
- VSAM online

Code database calls in the APS Program Painter for batch, report, or complex online programs, or in the Online Express Specification Editor for an Express program.

IDMS DB

The APS/IDMS calls include the following:

<i>DB-BIND</i>	Bind all records and the run-unit.
<i>DB-CLOSE</i>	Close a record.
<i>DB-ERASE</i>	Delete a record.
<i>DB-GET</i>	Move data into Working-Storage.
<i>DB-OBTAIN</i>	Read a record.
<i>DB-OPEN</i>	Open a record.
<i>DB-MODIFY</i>	Update a record.
<i>DB-STORE</i>	Add a record.
<i>DB-PROCESS</i>	Obtain and loop records.

<i>IDM-COMMIT</i>	Write a commit checkpoint.
<i>IDM-CONNECT</i>	Connect a record to a set.
<i>IDM-DISCONNECT</i>	Disconnect a record from set.
<i>IDM-IF</i>	Test a conditional.
<i>IDM-PROTOCOL</i>	Specify the program execution mode.
<i>IDM-RETURN</i>	Return the database key from an indexed set.
<i>IDM-ROLLBACK</i>	Write an abort checkpoint to an IDMS journal file.

IMS DB

The APS/IMS DB calls include the following:

<i>DB-ERASE</i>	Delete a record.
<i>DB-MODIFY</i>	Update a record.
<i>DB-OBTAIN</i>	Read a record.
<i>DB-PROCESS</i>	Read multiple records in a loop.
<i>DB-STORE</i>	Add a record.
<i>\$IM-FSA</i>	Build a Field Search Argument.
<i>\$IM-POS</i>	Access Data Entry Data Bases.

SQL

The APS/SQL DB calls include the following:

<i>DB-CLOSE</i>	Close a cursor set.
<i>DB-COMMIT</i>	Perform the SQL COMMIT function.
<i>DB-DECLARE</i>	Declare a cursor set.
<i>DB-ERASE</i>	Delete a row.
<i>DB-FETCH</i>	Obtain a row from a cursor set.
<i>DB-MODIFY</i>	Update a row.
<i>DB-OBTAIN</i>	Read a table.
<i>DB-OPEN</i>	Open a cursor set.
<i>DB-PROCESS</i>	Read a table and loop on it.
<i>DB-ROLLBACK</i>	Perform the SQL ROLLBACK function.
<i>DB-STORE</i>	Add a record.

VSAM Online

The APS/VSAM online DB calls include the following:

<i>DB-ERASE</i>	Delete a record.
<i>DB-FREE</i>	Release file resources.
<i>DB-MODIFY</i>	Update a record.
<i>DB-OBTAIN</i>	Read a file.
<i>DB-PROCESS</i>	Read a file and loop on it.
<i>DB-STORE</i>	Add a record.

VSAM Batch

The APS/VSAM batch DB calls include the following:

<i>DB-CLOSE</i>	Close a file.
<i>DB-ERASE</i>	Delete a record.
<i>DB-MODIFY</i>	Update a record.
<i>DB-OBTAIN</i>	Read a file.
<i>DB-OPEN</i>	Open a file.
<i>DB-PROCESS</i>	Read a file and loop on it.
<i>DB-STORE</i>	Add a record.

Coding Conventions

APS Logical View DB calls can consist of a call name, keywords, and arguments. Observe the following conventions when you code a DB call:

- Code the call using COBOL indentation. Improperly indented calls may cause errors.
- Argument values can be:
 - Variables
 - 1- to 7-digit numbers
 - Literal strings delimited by single or double quotation marks
- Separate each call component with a space, unless indicated otherwise in the syntax.

- Never code comments within database calls. To code comments, see *Comments*.
- Continue a call on as many as 101 subsequent lines by coding an ellipsis followed by a space (...). Break a call for continuation at any blank space, but do not break a parameter.
- Do not extend the call past column 72.

Related Topics:	See...	For more information about...
	<i>Error Handling</i>	Using IDMS database calls in program logic
	<i>ID Parameters</i>	
	<i>IDMS</i>	
	<i>IDMS DB Sample Programs</i>	
	<i>CCODE</i>	Using IMS DB database calls in program logic
	<i>Error Handling</i>	
	<i>GSAM Calls</i>	
	<i>SUPPRESS (IMS DB Option)</i>	
	<i>System Service Calls</i>	Using VSAM Batch database calls in program logic
	<i>Error Handling</i>	
	<i>Fields and Flags, Data Communication</i>	
	<i>Variable Length File Support</i>	
	<i>Error Handling</i>	Using VSAM Online database calls in program logic
	<i>Fields and Flags, Data Communication</i>	
	<i>Error Handling</i>	
	<i>Expressions, SQL</i>	
	<i>Functions, SQL</i>	Using SQL database calls in program logic
	<i>GROUP BY</i>	
	<i>Joins</i>	
	<i>NULL Indicator Field</i>	
	<i>Special Registers</i>	
	<i>Subselect Clause</i>	
	<i>UNION</i>	

See...	For more information about...
<i>Control Files</i>	Controlling certain target-specific functions
<i>Comments</i>	Entering comments in your program
<i>Reserved Words</i>	Avoiding use of APS reserved words
<i>Limits</i>	Recognizing size limitations

Data Communication Calls

Compatibility: All targets

Description: APS Logical View Data Communication (DC) calls are predefined, easy-to-use statements that let you focus on what needs to be accomplished, rather than the mechanics of the target environment. To address environment-specific requirements, you can extend these calls with keywords; no native coding is required. These calls facilitate both generic processing and specialized requests.

APS supports the following data communication targets:

- CICS
- IMS DC
- ISPF Dialog (DLG)
- ISPF prototyping

Code data communication calls in the APS Program Painter for batch, report, or complex online programs, or in the Online Express Specification Editor for an Express program.

*List of DC Calls***CICS**

The APS/CICS DC calls include the following:

<i>ATTR</i>	Override default I/O screen field attributes at run time.
<i>CIC-ADDRESS</i>	Access CICS storage areas.
<i>CIC-ASSIGN</i>	Assign values defined outside the program to a data area in Working-Storage.
<i>CIC-CANCEL</i>	Cancel a CIC-START or CIC-DELAY.
<i>CIC-DELAY</i>	Suspend a task.
<i>CIC-DELETEQ-TD</i>	Delete all transient data in a transient data queue.
<i>CIC-DELETEQ-TS</i>	Delete temporary data in a temporary storage queue and free all storage in the queue.
<i>CIC-FREEMAIN</i>	Release storage acquired by a CIC-GETMAIN call.
<i>CIC-GETMAIN</i>	Obtain and initialize main storage.
<i>CIC-LOAD</i>	Load programs, tables, or maps from a resident system library to main storage.
<i>CIC-READQ-TD</i>	Read transient data from a transient data queue.
<i>CIC-DELETEQ-TS</i>	Read a temporary storage queue in main or auxiliary storage.
<i>CIC-RELEASE</i>	Delete from main storage any programs, tables, or maps loaded by CIC-LOAD.
<i>CIC-SCHEDULE-PSB</i>	Schedule an IMS PSB.
<i>CIC-SEND-TEXT</i>	Clear the screen and unlock the keyboard before terminating a program.
<i>CIC-SERVICE-RELOAD</i>	Establish addressability to a data area in the Linkage Section following an address change in its BLL cell.
<i>CIC-START</i>	Start a task on a local or remote system.
<i>CIC-TERM-PSB</i>	Terminate the currently scheduled IMS PSB.
<i>CIC-WRITEQ-TD</i>	Write transient data to a predefined data queue.
<i>CIC-WRITEQ-TS</i>	Write temporary records to a temporary storage queue.
<i>CLEAR</i>	Move spaces to all screen fields.

<i>CLEAR-ATTRS</i>	Reset screen field attributes to their original values.
<i>LINK</i>	Transfer control to a subprogram and optionally send Commarea data.
<i>RESET-PFKEY</i>	Simulate screen invocation.
<i>SCRNLIST</i>	Enable the program to receive multiple screens.
<i>SEND</i>	Send a screen to the monitor.
<i>TERM</i>	Terminate the program.
<i>TP-BACKOUT</i>	ABEND the program.
<i>TP-COMMAREA</i>	Generate a Working-Storage record for data that the program can send to and receive from other programs.
<i>TP-LINKAGE</i>	Handle addressability of Linkage Section records.
<i>TP-NULL</i>	Move LOW-VALUES to all fields of a specified screen.
<i>TP-PERFORM</i>	Perform a paragraph and optionally pass arguments.
<i>XCTL</i>	Transfer program control to another program at the same logical level, and send Commarea data.

IMS DC

The APS/IMS DC calls include the following:

<i>ATTR</i>	Override default I/O screen field attributes at run time.
<i>CLEAR</i>	Move spaces to all screen fields.
<i>CLEAR-ATTRS</i>	Reset screen field attributes to their original values.
<i>\$IM-CHNG</i>	Issue a change call to another IO PCB.
<i>\$IM-CMD</i>	
<i>\$IM-GCMD</i>	
<i>\$IM-GN</i>	Issue a read of the next IMS message.
<i>\$IM-GU</i>	Issue a unique read of an IMS message.
<i>\$IM-ISRT</i>	Insert an IMS message.
<i>\$IM-PURG</i>	Issue a purge for a PCB.

<i>LINK</i>	Transfer control to a subprogram and optionally send Commarea data.
<i>MSG-SW</i>	Transfer control to another program and optionally send screen data.
<i>RESET-PFKEY</i>	Simulate screen invocation.
<i>SCRNLIST</i>	Enable the program to receive multiple screens.
<i>SEND</i>	Send a screen to the monitor.
<i>TERM</i>	Terminate the program.
<i>TP-BACKOUT</i>	ABEND the program.
<i>TP-COMMAREA</i>	Generate a Working-Storage record for data that the program can send to and receive from other programs.
<i>TP-LINKAGE</i>	Handle addressability of Linkage Section records.
<i>TP-NULL</i>	Move LOW-VALUES to all fields of a specified screen.
<i>TP-PERFORM</i>	Perform a paragraph and optionally pass arguments.

ISPF Dialog

The APS/ISPF Dialog DC calls include the following:

<i>ATTR</i>	Override default I/O screen field attributes at run time.
<i>CLEAR</i>	Move spaces to all screen fields.
<i>CLEAR-ATTRS</i>	Reset screen field attributes to their original values.
<i>DLG-ISPEXEC</i>	Invoke services through ISPEXEC calls.
<i>DLG-ISREDIT</i>	Invoke services through ISREDIT calls.
<i>DLG-SETMSG</i>	Display a message on the next panel.
<i>DLG-VCOPY</i>	Copy an ISPF Dialog variable value to a COBOL variable.
<i>DLG-VDEFINE</i>	Link an ISPF Dialog variable and a COBOL variable.
<i>DLG-VDELETE</i>	Delete an ISPF Dialog variable from the function pool.
<i>DLG-VREPLACE</i>	Move a COBOL variable value to the function pool.

<i>DLG-VRESET</i>	Reset function pool variables.
<i>LINK</i>	Transfer control to a subprogram and optionally send Commarea data.
<i>RESET-PFKEY</i>	Simulate screen invocation.
<i>SCRNLIST</i>	Enable the program to receive multiple screens.
<i>SEND</i>	Send a screen to the monitor.
<i>TERM</i>	Terminate the program.
<i>TP-COMMAREA</i>	Generate a Working-Storage record for data that the program can send to and receive from other programs.
<i>TP-LINKAGE</i>	Handle addressability of Linkage Section records.
<i>TP-NULL</i>	Move LOW-VALUES to all fields of a specified screen.
<i>TP-PERFORM</i>	Perform a paragraph and optionally pass arguments.
<i>XCTL</i>	Execute the LINK call.

ISPF Prototyping

The APS/ISPF prototyping DC calls include the following:

<i>ATTR</i>	Override default I/O screen field attributes at run time.
<i>LINK</i>	Transfer control to a subprogram and optionally sends Commarea data.
<i>MSG-SW</i>	Transfer control to another program and optionally send screen data.
See <i>RESET-PFKEY</i>	Simulate screen invocation.
<i>SCRNLIST</i>	Enable the program to receive multiple screens.
<i>SEND</i>	Send a screen to the monitor.
<i>TERM</i>	Terminate the program.
<i>TP-COMMAREA</i>	Generate a Linkage Section record for data that the program can send to and receive from other programs.
See <i>TP-PERFORM</i>	Perform a paragraph and optionally pass arguments.

XCTL Transfer program control to another program at the same logical level, and send Commarea data.

Coding Conventions

APS Logical View DC calls can consist of a call name, keywords, and arguments. Observe the following conventions when you code a DC call:

- Code the call using COBOL indentation. Improperly indented calls may cause errors.
- Argument values can be:
 - Variables
 - 1- to 7-digit numbers
 - Literal strings delimited by single or double quotation marks
- Separate each call component with a space, unless indicated otherwise in the syntax.
- Code positional arguments in the order shown in the syntax for each call.
- To omit a positional argument, code an asterisk (*) in its place, except for the last argument.
- Never code comments within data communication calls. To code comments, see *Comments*.
- Continue a call on as many as 101 subsequent lines by coding an ellipsis followed by a space (...). Break a call for continuation at any blank space, but do not break a parameter.
- Do not extend the call past column 72.

Related Topics	See...	For more information about...
	<i>Error Handling</i>	Using CICS data
	<i>PF Key Values</i>	communication calls in
	<i>Program Specification Blocks</i>	program logic
	<i>TP-COMMAREA</i>	

See...	For more information about...
<i>Error Handling</i>	Using IMS DC data communication calls in program logic
<i>\$IM- Data Communication Calls</i>	
<i>Program Control Blocks, IO</i>	
<i>System Service Calls</i>	
<i>TP-COMMAREA</i>	
<i>ISPF Dialog Compatibility with IMS DC, CICS</i>	Using ISPF Dialog data communication calls in program logic
<i>PF Key Values</i>	
<i>TP-COMMAREA</i>	
<i>Comments</i>	Entering comments in your program
<i>Control Files</i>	Controlling certain target-specific functions
<i>Reserved Words</i>	Avoiding use of APS reserved words
<i>Limits</i>	Recognizing size limitations

Data Structure Definition (DS01)

Category: APS-generated report (see *Application Reports*)

Description: The Data Structure Definition Reports displays data structure components exactly as painted, together with the following supplementary information.

- The Data Structure name and creation date
- The Data Structure title and date of last update

The report documents this aspect of your application to support future maintenance and enhancement efforts.

Comment: Produce the Data Structure Definition Report from the Report Generator, Painter Menu, or Application Painter.

Data Structures

Description: APS lets you code reusable data structures for programs and copy libraries, using a shorthand format in the Data Structure Painter. Or, you can code these data structures specifically for your program in the Program Painter. The shorthand format substitutes indentation for level numbers and allows shorthand picture formats. The APS Generators insert level numbers for indentation levels, expand the shorthand formats to full COBOL formats, and insert the necessary punctuation. Alternately, you can code data structures in the standard COBOL format.

Constructs:

<i>VALUE (Data Structure)</i>	Specify a VALUE clause.
<i>RED</i>	Specify a REDEFINES clause.
<i>OCCURS</i>	Specify an OCCURS clause.
<i>88</i>	Assign a value(s) to an 88-level variable.
<i>66 . . . RENAMES</i>	Designate a 66-level RENAMES clause.

Edit Mask Characters

Code the following COBOL edit mask characters in your data structures.

A P Z + - * B \$ 0

For example:

APS Code	Generated Code
A(12)000B	PIC A(12)000B.
PPP999	PIC PPP999.
ZZZ9.99	PIC ZZZ9.99
+999.99	PIC +999.99.
****.**	PIC ****.**.
S999PPP	PIC S999PPP.

Picture Formats

Code data structures in the Data Structure Painter format, which is a shorthand syntax.

Format	Generated COBOL
9	PIC 9.
99	PIC 99
999	PIC 999
9999	PIC 9999
9(<i>n</i>)	PIC 9(<i>n</i>)
9 <i>n</i>	PIC 9(<i>n</i>)
9& <i>variable</i>	PIC 9(& <i>variable</i>)
A	PIC A.
AA	PIC AA.
AAA	PIC AAA.
A(<i>n</i>)	PIC A(<i>n</i>).
A <i>n</i>	PIC A(<i>n</i>).
A& <i>variable</i>	PIC A(& <i>variable</i>).
C	PIC S9 COMP.
C(<i>n</i>)	PIC S9(<i>n</i>) COMP.
C <i>n</i>	PIC S9(<i>n</i>) COMP.
C-3	PIC S9 COMP-3.
C <i>n</i> -3	PIC S9(<i>n</i>) COMP-3.
C& <i>variable</i>	PIC S9(& <i>variable</i>) COMP.
C& <i>variable</i> +3	PIC S9(& <i>variable</i>) COMP-3.
F FULL	PIC S9(9) COMP.
H HALF	PIC S9(4) COMP.
I INDEX	INDEX. (For [USAGE IS] INDEX)
N	PIC 9.
N(<i>n</i>)	PIC 9(<i>n</i>).
N <i>n</i>	PIC 9(<i>n</i>).
N& <i>variable</i>	PIC 9(& <i>variable</i>).
P POINTER	POINTER. (For COBOL/2 only: [USAGE IS] POINTER.)
R REDEF REDEFINES	REDEFINES <i>dataname</i> .

Format	Generated COBOL
S	PIC S9.
S(n)	PIC S9(n).
Sn	PIC S9(n).
S&variable	PIC S9(&variable).
SYNC	SYNCHRONIZED
V(n)	PIC V9(n).
Vn	PIC V9(n).
V&variable	PIC V9(&variable).
Vvalueclause	VALUE valueclause.
X	PIC X.
XX	PIC XX.
XXX	PIC XXX.
XXXX	PIC XXXX.
Xn	PIC X(n).
X(n)	PIC X(n).
X&variable	PIC X(&variable).
...	Continuation of a format

- Comments:**
- Application generation treats most format specifications beginning with COBOL edit mask characters as a valid COBOL formats, and passes them directly to the generated program. However, APS translates an A, followed by a numeral, as follows.

A8 generates *PIC A(8)*.

- Enter only the level 66, 77, or 88 numbers; do not enter other level numbers, such as 01, 05.
- Enter one data element name per line only. A data element name can be a valid COBOL name of 1 - 30 characters, or expression introduced by one of the Customizer default symbols.
- You can continue the data structure on multiple lines wherever a space occurs, for example:

```

----- Data Structure Painter -----
WORK1-FIELD1      X(13)
... V'13 CHARS LONG'
WORK1-FIELD2      X(120)
... VALUE 'A LONG LITERAL MAY
... BE CONTINUED ON ONE OR
... MORE LINES'

```

- You do not need to code the word PICTURE in a picture clause. A PICTURE clause format can be a valid COBOL format or a APS Data Structure Painter shorthand picture format.
- If you do code the PICTURE or PIC, use only valid COBOL syntax within the PIC clause--do not use the APS shorthand picture format.
- If you code the word PIC, you must follow it with valid COBOL syntax.
- APS treats the Data Structure Painter formats 99, A9, N9, S9, and V9 as valid COBOL formats and passes them directly to the generated program (as PIC 99, PIC A9, and so on). If, however, you want the generated COBOL to be PIC 9(9), PIC V(9), and PIC A(9), use the Data Structure Painter formats 9(9), V(9), and A(9).
- If a format includes an Customizer variable followed by additional format syntax, use a plus sign (+) to separate the Customizer variable from the additional syntax. During processing, APS eliminates the plus sign and concatenates the suffix onto the syntax. For example,

`9&variable+V5` generates `PIC 9(&variable)V9(5)`.

- Distinguish *Vvalueclause* from V as an implied decimal point.
 - For *Vvalueclause*, precede the V with a space and immediately follow it with the valueclause.
 - For V as an implied decimal point, use no space before or after the V.

For example:

`N8V4` generates `PIC 9(8) V9(4)`.
`N8 V4` generates `PIC 9(8) VALUE 4`.

- Never code comments within Data Structure Painter constructs. To code comments, see *Comments*.

Example: The following illustrates using indentation to create level numbers.

Data Structure Painter code:

```
-LINE- ----- Data Structure Painter -----
000001          PROG-SPECIFIC-WORKING-DATA
000002          PGWS-FIELD1
000003          PGWS-FIELD2
000004          PGWS-FIELD3
```

```

000005                PGWS-FIELD4
000006                PGWS-FIELD5
000007                PGWS-FIELD6
000008                PGWS-FIELD7 X13

000009                ... V'ABCDEFGHijkl'
000010                PGWS-FIELD8
000011                PGWS-FIELD9
000012                PGWS-FIELD10 N13

```

Generated COBOL code:

```

01  PROG-SPECIFIC-WORKING-DATA.
05  PGWS-FIELD1.
10  PGWS-FIELD2.
15  PGWS-FIELD3.
20  PGWS-FIELD4.
25  PGWS-FIELD5.
30  PGWS-FIELD6.
35  PGWS-FIELD7
    PIC X(13)
    VALUE 'ABCDEFGHijkl'.
15  PGWS-FIELD8.
20  PGWS-FIELD9.
25  PGWS-FIELD10
    PIC 9(13).

```

Date and Time Field Edits

Category: Screen Painter feature (see *Field Edits*)

Description: Specify the storage format, the format and data requirements that the end user must adhere to when entering data into a date or time screen, and specify the format requirements for displaying the date or time.

Procedure: To assign internal, input, or output edits to a date screen, follow these steps.

- 1 Access the Screen Painter, then access the Field Edit Facility. From the Edit Selection window, select the **Special Edits** option.

- 2 From the Special Edits Screen Painter, select one of the following.

Option	Description
Date--Predefined Edits	Gregorian, Julian, or system date storage format and a predefined list of input/output formats
Date--User-Defined Edits	Storage, input, and output date format that you define
Time--User-Defined Edits	Storage, input, and output formats that you define

- 3 Assign date edits to fields as listed in the following Predefined Dates, User-Defined Dates, and Time Field tables or procedures.

Predefined Dates

Field	Description and Values
Storage Format	Specify the internal storage format as follows. <i>J</i> Julian--generates X(5) <i>JP</i> Julian packed--generates 9(5) COMP-3 <i>G</i> Gregorian--generates X(6) <i>GP</i> Gregorian packed--generates 9(6) COMP-3 If you do not specify a Storage Format, you must select the System Data Displayed option to capture the system date.
In/Out Format	Select the format. All formats are valid input formats. The field length determines if the special characters are assigned. For example, if you select the MM/DD/YY format, and the field length is 8, then APS assigns MM/DD/YY to the field; if the field length is 6, APS assigns MMDDYY to the field.
Date Required	Select to indicate that the end user must enter a value in the field.
Error Processing	Select to transfer to the Error Processing window to specify error messages and attributes when the field fails input edits.
System Data Displayed	Select to capture the system date. Select this option if you do not specify a storage format.

Field	Description and Values
Application Editing	Select to transfer to the Application Editing window to specify your own edits in a paragraph, subprogram, or APS rule for input or output.

User-Defined Dates

Field	Description and Values
Internal Picture	Select to transfer to the Internal Picture window to specify the COBOL picture characteristics.
Storage Format Input Format Output Format	<p>Type the internal storage format mask, input format mask, and output format mask. Valid mask characters are Y (year), M (month), D (day), and special characters if the field is not defined numeric. Restrictions are</p> <p><i>Y</i> Can be 2 or 4 characters, where:</p> <ul style="list-style-type: none"> • 2 Ys indicate a numeric year, such as 93. • 4 Ys indicate a numeric century and year, such as 1993. <p><i>M</i> Can be 2, 3, or 9 characters, where:</p> <ul style="list-style-type: none"> • 2 Ms indicate a numeric month, such as 12. • 3 Ms indicate a short character month, such as DEC. • 9 Ms indicate a long character month, such as DECEMBER. <p><i>D</i> Can be 3 characters if you do not define Month, otherwise it must be 2 characters, where:</p> <ul style="list-style-type: none"> • 2 Ds indicate a numeric Gregorian day, such as 30. • 3 Ds indicate a numeric Julian day, such as 360. <p>The storage length must equal the Internal Picture length. Month, day, and year must be in the same order in both the Input and Output Formats.</p>

Field	Description and Values
	<p>Example:</p> <p>Storage Format YYYYMMDD</p> <p>Input Format MMDDYY</p> <p>Output Format MMMMMMMMMM DD, YYYY</p>
Date Required	Select to indicate that the end user must enter a value in the field.
Error Processing	Select to transfer to the Error Processing window to specify error messages and attributes when the field fails input edits.
System Date Data	Select one of the following. <ul style="list-style-type: none"> I During input editing, insert the system date only if the field is blank. IR During input editing, always insert the system date, regardless of the field contents. O During output editing, insert the system date only if the field is blank. OR During output editing, always insert the system date, regardless of the field contents.
Application Editing	Select to transfer to the Application Editing window to specify your own edits in a paragraph, subprogram, or APS rule for input or output.

Time Fields

Field	Description
Internal Picture	Select to transfer to the Internal Picture window to specify the COBOL picture characteristics.
Storage Format	Specify the internal storage format mask. Valid characters are HH (hour), MM (minute), SS (second), *s and special characters if the field is not defined numeric. The storage format must equal the Internal Picture length. Seconds are optional. For example, HH:MM**

Field	Description
Input Format and Output Format	<p>Type the input and output format masks. Valid mask characters are HH (hour), MM (minute), SS (second), *s, and special characters if the field is not defined numeric. Seconds are optional.</p> <p>Append your format with asterisks to indicate the AM and PM indicators that you specify, using a one-to-one correspondence. For example, if your indicators are A.M. and P.M., you would append **** to your format; if they are AM and PM, you would append ** to your format.</p> <p>Hours, minutes, and seconds must be in the same order in both the Input and Output Formats. For example:</p> <p>Input Format HHMMSS Output Format HH.MM.SS**</p>
Input Required	Select to indicate that the end user must enter a time value in the field.
Error Processing	Select to transfer to the Error Processing window to specify error messages and attributes when the field fails input edits.
Application Editing	Select to transfer to the Application Editing window to specify your own edits in a paragraph, subprogram, or APS rule for input or output.
System Time Data	<p>Select one of the following:</p> <ul style="list-style-type: none"> <li data-bbox="711 1177 1282 1237">I During input editing, insert the system time only if the field is blank. <li data-bbox="711 1255 1282 1341">IR During input editing, always insert the system time, regardless of the field contents. <li data-bbox="711 1359 1282 1420">O During output editing, insert the system time only if the field is blank. <li data-bbox="711 1437 1282 1524">OR During output editing, always insert the system time, regardless of the field contents.

Field	Description
AM Indicator	Type the AM time indicator, such as a.m. or am.
PM Indicator	Type the PM time indicator, such as p.m. or pm.

Note: In a report mock-up, use PIC clauses instead of COBOL masks when formatting dates and times containing / \$ or :. See *Report Mock-Ups*.

DB/DC Target Combinations

Description: The following table shows all valid DB/DC combinations for generating executable programs.

DC Target	DB Target
CICS	DB2
	IMS
	VSAM
IMS	DB2
	IDMS
	IMS
ISPF	DB2
	VSAM
DLG (ISPF Dialog)	IDMS
	DB2
	VSAM
MVS (batch)	DB2

DC Target	DB Target
	IMS
	VSAM

DB-BIND

Category: Database call (see *Database Calls*)

Compatibility: IDMS target

Description: Bind all records copied into a program subschema via a COPY IDMS statement; bind the run unit.

Syntax: Format 1:

```
DB-BIND REC [recordname]
```

Format 2:

```
DB-BIND RUN-UNIT
... SUBSCHEMA name | NODENAME name | DBNAME name
```

Parameters:	DBNAME <i>name</i>	Database name from the IDMS database name table.
	NODENAME <i>name</i>	IDMS network node.
	REC [<i>recordname</i>]	Retrieve record. Recordname is optional because the record is previously located by a DB-OBTAIN REF.
	RUN-UNIT	Specify binding, if IDMS run unit.
	SUBSCHEMA <i>name</i>	Bind the run unit with the subschema.

- Comments:**
- All programs accessing IDMS databases must be bound to the run unit.
 - DB-BIND generates the IDMS command COPY IDMS SUBSCHEMA-BINDS.

- All programs with a subschema generate a DB-BIND, but if protocol is manual, you must code a DB-BIND REC recordname. Coding DB-BIND overrides the automatic generation of the call.
- If IDM-PROTOCOL MANUAL is not coded, do not code any options with DB-BIND. APS generates a COPY IDMS RECORD *recordname* for every DB-BIND REC *recordname*.
- No status flags are generated; paragraph IDMS-STATUS checks for errors.

Example:

```

-KYWD- 12-*----20---*-----30---*-----40---*-----50---*-----60
SYEN   IDM-PROTOCOL BATCH-AUTOSTATUS MANUAL
      .
      .
      .

PROC
      DB-BIND RUN-UNIT DBNAME TESTDB
      DB-BIND REC EMPLOYEE
      DB-OPEN MODE RETRIEVAL
      PERFORM 100-PROCESS-RTN

```

DB-CLOSE

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB, SQL, and VSAM batch targets

Description: Close IDMS files and subschema areas; close SQL cursor sets; close VSAM batch files.

Syntax: IDMS DB

```
DB-CLOSE ALL
```

VSAM Batch

Format 1:

```
DB-CLOSE FILE filename1 [ ... filenameN]
```

Format 2:

DB-CLOSE FILE ALL

SQL

DB-CLOSE CUR[SOR] *cursorname*

Parameters:	ALL	Specify all files and ready all areas defined in the subschema.
	CUR[SOR] <i>cursorname</i>	Specify cursor. <i>Cursorname</i> must be previously named by DB-DECLARE or DB-PROCESS-ID.
	FILE <i>filename</i>	File(s) to process.

Comments:

- IDMS DB**
- DB-CLOSE nullifies all currencies; execute appropriate DB-BIND and DB-OPEN calls in order to use the database again within the same program.
 - APS generates DB-CLOSE for all online programs unless the program is a linked program or DB-CLOSE is coded.

SQL

A cursor set can be opened, processed, and closed multiple times in the same program; you must code DB-CLOSE before you invoke another DB-OPEN.

VSAM Batch

DB-CLOSE is required.

DB-COMMIT

Category: Database call (see *Database Calls*)

Compatibility: SQL target

Description: Perform database commit functions.

Syntax: DB-COMMIT [HOLD]

Parameters: HOLD Do not release resources; do not close open cursors; preserve prepared SQL statements; release locks on specific rows acquired during the transaction.

Comments: Any SQL program running under ISPF prototype or ISPF Dialog generates DB2 COMMITs upon normal termination of the program (for example, through TP-SEND, TP-XCTL, TP-TERM). In addition, APS provides two variables you can code that generate COMMITs for DB calls under either ISPF prototyping or ISPF Dialog.

- &DB2-AUTO-COMMIT = 0. Code this variable before the NTRY call in your program to generate a COMMIT for the following DB calls.

DB-STORE

DB-ERASE (except with WHERE or WHERE CURRENT)

DB-MODIFY (except with WHERE or WHERE CURRENT)

- &TP-ISPF-DB2-COMMIT-NEEDED = 1. If you modify the NTRY call in such a way that you need a COMMIT generated at the end of your modification routine (for example, if you perform an update or open a cursor), code this variable and set it to 1 before the NTRY call in your program. You can modify NTRY by updating \$TP-ENTRY functions in the APS CNTL file APDLGIN and coding edit routines.

DB-DECLARE

Category: Database call (see *Database Calls*)

Compatibility: SQL target

Description: Designate a set of rows as a logical group, that is, a cursor set. The call declares:

- All rows and columns in a table
- All columns, from specific rows, in a table
- Specific columns in a table

- Specific columns, from specific rows, in a table

Syntax: Format 1, unqualified, select all columns:

```
DB-DECLARE cursorname copylibname-REC
... [FETCH ONLY]
... [WITH HOLD]
... [OPTIMIZE number]
... [UPDATE|ORDER]
... column1 [ASC|DESC] [...columnN [ASC|DESC]]
```

Format 2, qualified, select all columns:

```
DB-DECLARE cursorname copylibname-REC
... [FETCH ONLY]
... [WITH HOLD]
... [OPTIMIZE number]
... WHERE column operator [[:]altvalue]|column
... [AND|OR column operator [[:]altvalue]|column]
... [UPDATE|ORDER]
... column1 [ASC|DESC] [...columnN [ASC|DESC]]
```

Format 3, select specific columns:

```
DB-DECLARE cursorname copylibname-REC
... [DISTINCT]
... [FETCH ONLY]
... [WITH HOLD]
... [OPTIMIZE number]
... column1 [(altvalue)] [... columnN [(altvalue)]]
... [WHERE column operator [[:]altvalue]|column]
... [AND|OR correlname.]column operator [[:]altvalue]|column]
.
.
.
... [AND|OR correlname.]column operator [[:]altvalue]|column]]
... [UPDATE|ORDER]
... column1 [ASC|DESC] [...columnN [ASC|DESC]]
```

Format 4, join columns from two or more tables:

```
DB-DECLARE cursorname correlname.copylibname-REC
... [DISTINCT]
... [FETCH ONLY]
... [WITH HOLD]
... [OPTIMIZE number]
... [column1 [(altvalue)] [... columnN [(altvalue)]]]
... [correlname.copylibname-REC]
... [column1 [(altvalue)] [... columnN [(altvalue)]]]
```

```

.
.
.
... [WHERE correlname.column operator
[:]altvalue|correlname.column
... [AND|OR correlname.column oper
[:]altvalue|correlname.column]
.
.
.
... [AND|OR correlname.column operator
[:]altvalue|correlname.col]]
... [ORDER
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]

```

Format 5, specify a UNION:

```

DB-DECLARE cursorname copylibname-REC
... [DISTINCT]
... [FETCH ONLY]
... [WITH HOLD]
... [OPTIMIZE number]
... [column1 [(altvalue)] [... columnN [(altvalue)]]]
... [WHERE column operator [[:]altvalue]|column
... [AND|OR column operator [[:]altvalue]|column]
.
.
.
... [AND|OR column operator [[:]altvalue]|column]
... UNION [ALL]
DB-OBTAIN REC copylibname-REC
.
.
.
... [ORDER
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]

```

Parameters: [:]*(altvalue)*

Alternate value; can be a literal, column name, or host-variable, as follows.

- A host-variable is any COBOL data item referenced in your APS/SQL code; can a be data item generated automatically by APS/SQL to match a DB2 column name.

- An alternate host-variable is one you instruct APS/SQL to use instead of the automatically generated one for a column.

Precede host-variables and alternate host-variables with a colon. APS generates a # symbol for the colon.

*AND col op col[:]*altval**

Value can be a literal or data name. See also the *altvalue* parameter above.

copylibname-REC

Copybook library name of source data.

correlname.

Correlation name; maximum 18 characters, ending with a period. Required if columnname is in a select list or if the WHERE clause appears in multiple joined tables.

cursorname

Cursor name (maximum 12 characters) must be unique, and cannot be the same as the subschema copylib names.

DISTINCT

Eliminate all but one row from each set of duplicate rows. Duplicate rows have identical selected columns from the results table.

FETCH ONLY

Specify that the table is read-only and therefore the cursor cannot be referred to in positioned UPDATE and DELETE statements. Do not code in a call that contains an UPDATE clause.

OPTIMIZE number

Specify estimated maximum number of rows that call will retrieve. If the call retrieves no more than number rows, performance could be improved. Specifying this keyword does not prevent all rows from being retrieved.

*OR col op col[:]*altval**

Value can be a literal or data name. See also the *altvalue* parameter above.

ORDER [ASC DESC] [<i>col1 colN</i>]	Sort the results table in ascending (default) or descending order, based on the values in the columns specified. Specify the column either by name or by relative position in the column selection list. Specify at least one column. Do not code with UPDATE.
UPDATE <i>col1 colN</i>	Modify columns during cursor processing. In cursor processing, you cannot modify a column unless you code UPDATE first. Do not code UPDATE with UNION, DISTINCT, GROUP BY, or if call specifies a join or selects column functions.
WHERE <i>col op [:]altval</i>	Column is the column on which to qualify the selection. Operator can be =, ^=, >, <, >=, <=, native SQL predicates (such as LIKE and BETWEEN). See also the <i>altvalue</i> parameter above.
WITH HOLD	Prevent the closing of a cursor as a consequence of a commit operation. See also "Comments" below.

- Comments:**
- Declare a cursor before coding a DB-OPEN or DB-FETCH call.
 - Because a declared cursor name is referenced by all subsequent calls for that cursor, code UPDATE to specify which columns can be modified; otherwise columns cannot be modified in subsequent cursor processing.
 - When specifying columns for sorting, identify them either by name or position in the selection list; do not mix references.
 - When you code WITH HOLD, a commit operation commits all the changes in the current unit of work, but releases only locks that are not required to maintain the cursor. Afterwards, you must code an initial DB-FETCH before you can execute a positioned update or delete. After the initial DB-FETCH, the cursor is positioned on the row following the one it was positioned on before the commit operation.
 - The WITH HOLD clause is ignored in CICS and IMS DC.

Examples: Declare cursor set D2MAST-CURSOR; define to include all rows and columns in D2MASTER table; allow updating for PM_COLOR and PM_NEW_PART_NO columns.

```
DB-DECLARE D2MAST-CURSOR D2TAB-REC
... UPDATE PM_COLOR PM_NEW_PART_NO
```

Declare cursor set consisting of entire rows selected by evaluating two columns; if duplicate rows, select only one row where PM_PART_SHORT_DESC equals Working-Storage variable WS_PART_SHORT_DESC and PM_UNIT_BASE_PRICE is greater than 10 and less than 50.

```
DB-DECLARE D2MAST-CURSOR D2TAB-REC
... DISTINCT
... WHERE PM_PART_SHORT_DESC =
... :WS-PART-SHORT-DESC
... AND PM_UNIT_BASE_PRICE BETWEEN 10 AND 50
```

Declare cursor and define its set to include two columns; select only rows that meet selection criteria; for column PM_PART_NO, move data to default destination; for column PM_COLOR, move data to alternate host-variable WS-COLOR.

```
DB-DECLARE D2MAST-CURSOR D2TAB-REC
... PM_PART_NO
... PM_COLOR (WS-COLOR)
... WHERE PM_PART_SHORT_DESC =
... :WS-PART-SHORT-DESC
... AND PM_UNIT_BASE_PRICE BETWEEN 10 AND 50
```

Declare cursor and define its set to include two columns; select only one row that meets selection criteria. Sort columns by position within selection list; sort cursor first by PM_COLOR (second column in selection list), then within PM_COLOR by PM_PART_NO (first column).

```
DB-DECLARE D2MAST-CURSOR D2TAB-REC
... DISTINCT
... PM_PART_NO PM_COLOR
... WHERE PM_PART_SHORT_DESC =
... :WS-PART-SHORT-DESC
... ORDER 2, 1
```

Declare a cursor set to include three columns drawn from two separate select statements, UNIONed together. Select two columns from D2TAB-REC and one column from D2INVEN-REC in the first UNION. Base selection criteria on PM_PART_NO matching IN_PART_NO, with

PM_COLOR equal to the Working-Storage field WS-COLOR and IN-COLOR equal to 'BLUE'.

Select three columns from D2TAB-REC in the second UNION. Base selection criteria on PM_COLOR not equal to 'BLUE' and PM_UNITS less than 50. Sort the result table (consisting of rows returned from both select statements) by PM_PART_NO within PM_COLOR.

```
DB-DECLARE JOIN-CUR A.D2TAB-REC
... DISTINCT
... PM_PART_NO PM_COLOR
... B.D2INVEN_REC IN_QTY_ONHAND
... WHERE A.PM_PART_NO = B.IN_PART_NO
... AND A.PM_COLOR = :WS-COLOR
... AND B.IN_COLOR = 'BLUE'
... UNION
DB-OBTAIN REC D2TAB-REC
... PM_PART_NO PM_COLOR PM_UNITS
... WHERE PM_COLOR ^= 'BLUE'
... AND PM_UNITS < 50
... ORDER 1 ASC, 2 ASC
```

DB-ERASE

Category: Database call (see *Database Calls*)

Description: Delete a record or records.

Under IDMS, the call deletes:

- Record from all sets in which it participates as a member
- Record from database
- Mandatory and optional member records

Under IMS, the call deletes:

- Record and all dependent segments
- Record obtained by a path call DB-OBTAIN

Under SQL, the call deletes:

- All rows in a table
- Specific row in a table
- Cursor set rows

Under VSAM, the call deletes:

- Record(s) specified by key qualification via DB-OBTAIN (key qualified)
- Record retrieved by DB-OBTAIN or DB-PROCESS (unqualified)

Syntax: IDMS DB

```
DB-ERASE REC recordname [PERM|SELECT|ALL]
```

IMS DB

Format 1:

```
DB-ERASE REC segment [FROM dataarea]  
... [VIEW pcbname|PCB pcbname]
```

Format 2:, records obtained by path calls:

```
DB-ERASE REC|REF segment1 [FROM dataarea]  
... [VIEW pcbname|PCB pcbname]  
... REC|REF segment2 [FROM dataarea]  
... [VIEW pcbname|PCB pcbname]  
.  
.  
.  
... REC segmentN [FROM dataarea]  
... [VIEW pcbname|PCB pcbname]
```

SQL

Format 1:

```
DB-ERASE REC copylibname-REC  
... [WHERE column operator [:]altvalue  
... [AND|OR column operator [:]altvalue  
.  
.  
.  
... AND|OR column operator [:]altvalue]
```

Format 2:

```
DB-ERASE REC copylibname-REC
... [WHERE CURRENT [OF] cursorname]
```

VSAM Batch**Format 1, key-qualified:**

```
DB-ERASE REC recordname
... WHERE primarykeyname = value
... [SUB value] [OF dataarea]
```

Format 2, unqualified:

```
DB-ERASE REC recordname
```

VSAM Online**Format 1, key-qualified:**

```
DB-ERASE REC recordname
... WHERE primarykeyname = value
... [SUB value] [OF dataarea] [KLEN value]
... [SYSID systemname] [DDN ddname]
```

Format 2, unqualified:

```
DB-ERASE REC recordname
```

Parameters: [:](*altvalue*)

Alternate value; can be a literal, column name, or host-variable, as follows.

- A host-variable is any COBOL data item referenced in your APS/SQL code; can be a data item generated automatically by APS/SQL to match a DB2 column name.
- An alternate host-variable is one you instruct APS/SQL to use instead of the automatically generated one for a column.
- Precede host-variables and alternate host-variables with a colon. APS generates a # symbol for the colon.

ALL	Same as PERMANENT, but delete all optional records.
AND <i>col op [:]altval</i>	Value can be a literal or data name. See also the <i>altvalue</i> parameter above.
DDN <i>ddname</i>	File data description name; supply a value to the name option of CICS DATASET option. <i>Ddname</i> can be a literal or data name defined as PIC X(8).
FROM <i>dataarea</i>	Alternate I/O area where program deletes, modifies, or adds a record. Required for a record obtained from an I/O area other than the default I/O area, such as by DB-OBTAIN INTO. See also "Comments" below.
IMSREC <i>segmentname</i>	Specify that the segment name (maximum 8 characters) is in a Working-Storage variable for the program to read, modify, add, or delete.
KLEN <i>value</i> or KEYLENGTH <i>value</i>	Specify number or characters in key length; full or partial length is valid. Value can be a number or a data name defined as PIC S9(4) COMP. APS generates the CICS GENERIC option for a partial key length.
OF <i>dataarea</i>	Qualify the I/O area moving to the value field, when more than one structure in the Data Division contains the field. Optionally code IN instead of OF.
OR <i>col op [:]altval</i>	Value can be a literal or data name. See also the <i>altvalue</i> parameter above.
PCB <i>pcbname</i>	Synonymous with VIEW. Specify PCB used when PSB contains multiple PCBs for the same database.

PERM[ANENT]	<p>Disconnect optional records and delete</p> <ul style="list-style-type: none"> • Named record • Mandatory member records owned by the named record • Mandatory member records, if a mandatory member record is an owner
SELECT[IVE]	Same as PERMANENT, but delete optional records if they are not members of another set.
SUB[SCRIPT] (<i>value</i>)	Move the subscripted field value to a specified field. Value can be a data name, literal, or, under VSAM Batch or Online, an integer.
SYSID <i>systemname</i>	Process records stored on remote systems. Name a file residing in a remote data set. <i>Systemname</i> can be a 4-character literal region name or a Working-Storage field containing a 4-character region name.
VIEW <i>pcbname</i>	Synonymous with PCB. Specify the PCB used when the PSB contains multiple PCBs for the same database. See also "Comments" below.
WHERE <i>col op [:]altval</i>	Not valid for cursor processing. Column is the column on which to qualify the selection. Operator can be =, ^=, >, <, >=, <=, native SQL predicates (such as LIKE and BETWEEN). See also the <i>altvalue</i> parameter above.
WHERE <i>primarykey = val</i>	Value can be literal, data name, or an asterisk (*). An asterisk indicates the segment record description contains the key value.
WHERE CURRENT [OF] <i>cursor</i>	Valid for cursor processing only. Act upon the row retrieved from cursor.

Comments: IDMS DB

Record to be deleted must be current of run unit.

IMS DB

- DB-ERASE assumes record retrieval from the default I/O area. If this is not true, you must code FROM *dataarea*.
- When a record is obtained from a PCB other than the default PCB, such as by DB-OBTAIN with VIEW or PCB, you must code VIEW *pcbname*.
- Specify every segment down to the first segment being deleted for records obtained by path calls. That is, every record named in an DB-OBTAIN preceding an ERASE must be named in the DB-ERASE, also.

Examples: IDMS DB

Record type ORDER is current of run unit. Delete it from the database; disconnect it from all set occurrences in which the record participates; delete all mandatory and optional member records.

```
DB-ERASE REC ORDER ALL
```

IMS DB

Delete only RECORD-B and RECORD-C. First, issue the following DB-OBTAIN call.

```
DB-OBTAIN REC RECORD-A WHERE KEY-A = VALUE-A
... REC RECORD-B WHERE KEY-B = VALUE-B
... REC RECORD-C WHERE KEY-C = VALUE-C HOLD
```

Then, name all segments accessed by the prior path call DB-OBTAIN. Because the children of a deleted segment are also deleted, there is no need to code beyond the highest level segment being deleted (RECORD-B).

```
DB-ERASE REF RECORD-A
... REC RECORD-B
... REC RECORD-C
```

SQL

Delete any row in table D2MASTER where PM_PART_NO equals 123 or 567.

```
DB-ERASE REC D2TAB-REC
... WHERE PM_PART_NO = '123'
... OR PM_PART_NO = '567'
```

VSAM Batch

Delete a record where ORDR-NUMBER equals the value in the Working-Storage variable CUST-ORDR-NUMBER.

```
DB-ERASE REC ORDR-RECORD
... WHERE ORDR-NUMBER = CUST-ORDR-NUMBER
```

Read a CUST-RECORD for deletion.

```
DB-OBTAIN REC CUST-RECORD
... WHERE CUST-KEY = SCREEN-CUST-KEY
IF OK-ON-REC
    DB-ERASE REC CUST-RECORD
```

VSAM Online

Delete a group of ORDR-RECORD records; use partial key length. For successful deletes, store the number of actual records in APS data field APS-VSAM-NUMREC.

```
DB-ERASE REC ORDR-RECORD
... WHERE ORDR-NUMBER = SCREEN-PARTIAL-ORDR-NUMBER
... KLEN 6
IF OK-ON-REC
    SCREEN-MSG =
... 'ORDER RECORDS DELETED, NUMBER ORDERS = '
    SCREEN-NBR-RECS = APS-VSAM-NUMREC
```

Hold a CUST-RECORD for deletion.

```
DB-OBTAIN REC CUST-RECORD
... WHERE CUST-KEY = SCREEN-CUST-KEY
... HOLD
IF OK-ON-REC
    DB-ERASE REC CUST-RECORD
```

DB-FETCH

Category: Database call (see *Database Calls*)

Compatibility: SQL target

Description: Sequentially retrieve rows from the cursor set defined by DB-DECLARE.

Syntax: DB-FETCH CUR[SOR] *cursorname*
 ... [INTO *dataname*]

Parameters: CUR[SOR] *cursorname* Specify cursor. Cursorname must be previously named by DB-DECLARE or DB-PROCESS-ID.

INTO *dataname* Move host variable structure into the alternate data structure data name. Data moves after the actual SQL call via a MOVE statement. Generated code is:

```
IF OK-ON-REC
  MOVE hostname TO dataname
```

- Comments:**
- When you define a cursor set with DB-DECLARE, SQL places the selected rows in a results table, or cursor set. DB-FETCH retrieves these rows.
 - Each DB-FETCH returns the next row of the cursor set until the end of the results table is reached. The row returned by the current iteration of DB-FETCH is the current row.
 - You can code DB-FETCH within an S-COBOL loop to retrieve multiple rows from a cursor set.
 - To FETCH into individual columns, specify those alternate host variables in DB-DECLARE.

Example:

- Retrieve columns and rows in cursor set D2MAST-CURSOR; place information into WS-D2MAST-RECORD in Working-Storage.

```
DB-FETCH CURSOR D2MAST-CURSOR
... INTO WS-D2MAST-RECORD
```

DB-FREE

Category: Database call (see *Database Calls*)

Compatibility: VSAM online target

Description: Release file resources and:

- End a sequential DB-OBTAIN browse.
- Unlock a held record (a record held by a DB-OBTAIN HOLD) when DB-OBTAIN is not followed by DB-MODIFY or DB-ERASE.

Syntax: DB-FREE REC *recordname* | ALL
 ... [VIEW *keyname*]
 ... [ENDBR] [UNLOCK] [REQID *number*]
 ... [SYSID *systemname*] [DDN *ddname*]

Parameters: ALL	Process all subschema records. When coded with SYSID, all files must reside on the same region.
DDN <i>ddname</i>	Specify file <i>ddname</i> ; can be a literal or data name defined as PIC X(8). Supply a value to the name option of CICS DATASET.
ENDBR	End active browse; generate CICS ENDBR command. See also "Comments" below.
REC <i>recordname</i>	COBOL record to process.
REQID <i>number</i>	Unique browse identifier for performing a simultaneous browse on the same key; is a single integer (0 - 9). Assign &VS-ENDBR-CONTROL = "USER" in the APS CNTL file APVSAMIN.
SYSID <i>systemname</i>	Remote system name (maximum 4 characters); can be a literal region name or a Working-Storage field.
UNLOCK	Unlock file; generate CICS UNLOCK command. See also "Comments" below.
VIEW <i>keyname</i>	Specify primary or alternate key.

- Comments:**
- UNLOCK and ENDBR are default keywords. De-activate as follows.
 - Coding only ENDBR makes UNLOCK inactive.
 - Coding only UNLOCK makes ENDBR inactive.
 - When ALL is coded and UNLOCK and ENDBR are active, a single DB-FREE call (located in a central paragraph) works for the entire program. Note Code ALL only when &VS-ENDBR-CONTROL = "APS" (found in the APS CNTL file APVSAMIN).

Examples: Release a record held by a direct DB-OBTAIN; deactivate ENDBR.

```
DB-OBTAIN REC ORDR-RECORD
... WHERE ORDR-NUMBER = SCREEN-ORDR-NUMBER
... HOLD
DB-FREE REC ORDR-RECORD UNLOCK
```

Terminate a sequential DB-OBTAIN; specify the key used; deactivate UNLOCK.

```
DB-OBTAIN REC CUST-RECORD
... VIEW CUST-NUMBER
DB-FREE REC CUST-RECORD
... VIEW CUST-NUMBER ENDBR
```

DB-GET

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB target

Description: Move data from a previously located record (via a DB-OBTAIN REF) into Working-Storage.

Syntax: DB-GET REC [*recordname*]

Parameter: *recordname* Retrieve record. Recordname is optional because the record is previously located by a DB-OBTAIN REF.

Example: Record type ORDER is current of record type. After a successful DB-GET make ORDER data available to the program Working-Storage.

```
DB-GET REC ORDER
```

DB-MODIFY

Category: Database call (see *Database Calls*)

Description: Modify a record.

Under IDMS DB, the call rewrites the object record in the database from Working-Storage.

Under SQL, the call updates row contents in a table or cursor set, as follows.

- All rows in a table
- Specific rows in a table
- Specific columns in a table
- Specific columns of a specific row
- Cursor set rows

Syntax: **IDMS DB**

```
DB-MODIFY REC recordname
```

IMS DB

```
DB-MODIFY
... REC|REF recordname1 [FROM dataarea]
... [VIEW pcbname|PCB pcbname]
.
.
.
... REC|REF recordnameN [FROM dataarea]
... [VIEW pcbname|PCB pcbname]
```

SQL

```
DB-MODIFY REC copylibname-REC
... [column1 [(altvalue)] [... columnN [(altvalue)]]]
... [FROM dataname]
... [WHERE column1 operator [:]altvalue]
.
.
.
```

```
... AND|OR columnN operator [:]altvalue]
... [END[WHERE]]]
... [WHERE CURRENT [OF] cursorname]
```

VSAM Batch

```
DB-MODIFY REC recordname [FROM dataarea]
```

VSAM Online

```
DB-MODIFY REC recordname [FROM dataarea]
... [SYSID systemname] [DDN ddname]
```

Parameters:	[:](altvalue)	<p>Alternate value; can be a literal, column name, or host-variable, as follows.</p> <ul style="list-style-type: none"> • A host-variable is any COBOL data item referenced in APS/SQL code; can be a data item generated by APS/SQL to match DB2 column name. • Instruct APS/SQL to use an alternate variable instead of the variable automatically generated for a column. If you name alternate host-variables for specific columns, do not code FROM, which names an entire alternate host structure. • Precede host-variables and alternate host-variables with a colon. APS generates a # symbol for the colon.
	AND col op [:]altval	Value can be a literal or data name. See also the altvalue parameter above.
	DDN ddname	Specify file ddname; can be a literal or data name defined as PIC X(8). Supply a value to the name option of CICS DATASET.
	END[WHERE]	Terminate a WHERE clause. Required if you code WHERE before FROM.

FROM <i>dataarea</i>	Alternate I/O area where program deletes, modifies, or adds a record. Required for a record obtained from an I/O area other than the default I/O area, such as by DB-OBTAIN INTO.
FROM <i>dataname</i>	Move alternate data structure to the host variable structure name. Data moves prior to the actual SQL call via MOVE statement. Preferred format is to code FROM before WHERE, otherwise you must separate the WHERE and FROM with ENDWHERE. See also "Comments" below.
IMSREC <i>segmentname</i>	Specify that the segment name (maximum 8 characters) is in a Working-Storage variable for the program to read, modify, add, or delete.
OR <i>col op</i> [:] <i>altval</i>	Value can be a literal or data name. See also the <i>altvalue</i> parameter above.
PCB <i>pcbname</i>	Synonymous with VIEW. Specify the PCB used when the PSB contains multiple PCBs for the same database.
REC <i>copylib-REC</i>	Specify the 01-level name of the COBOL row layout in the DCLGEN or copybook information. Cannot be the same as any cursor names or DB-PROCESS-ID names.
REC <i>recordname</i>	COBOL record or IMS segment to process.
REF <i>recordname</i>	Specify a COBOL record to reference. Under IMS, the program uses the referenced segment for navigating the database. See also "Comments" below.
SYSID <i>systemname</i>	Remote system name (maximum 4 characters); can be a literal region name or a Working-Storage field.
VIEW <i>pcbname</i>	Synonymous with PCB. Specify the PCB used when the PSB contains multiple PCBs for the same database. See also "Comments" below.

WHERE *col*
op [:]*altval*

Not valid for cursor processing. Column is the column on which to qualify the selection. Operator can be =, ^=, >, <, >=, <=, native SQL predicates (such as LIKE and BETWEEN). See also the *altvalue* parameter above.

WHERE CURRENT [OF] *cursor* Valid for cursor processing only. Act upon the row retrieved from cursor.

Comments: IDMS DB

The object record must be current of run unit. If a CALC key is modified, the object record can be accessed by the new CALC key value.

IMS DB

- The record must be retrieved and held by a DB-OBTAIN or DB-PROCESS call before it can be modified.
- DB-MODIFY assumes record retrieval is from the default I/O area. If it is not, code FROM *dataarea*.
- When a record is not obtained from the default PCB, such as by DB-OBTAIN with VIEW or PCB, code VIEW *pcbname*.
- To modify records obtained by a path call DB-OBTAIN, specify every segment obtained in the path. To specify that a segment is not modified, code REF.

VSAM Batch and Online

Before modifying a record, retrieve it with a DB-OBTAIN or DB-PROCESS call.

Examples: IDMS DB

Rewrite record type ORDER, which is current of run unit, from Working-Storage.

```
DB-MODIFY REC ORDER
```

IMS DB

Modify only RECORD-C.

```
DB-OBTAIN REC RECORD-A WHERE KEY-A = VALUE-A
... REC RECORD-B WHERE KEY-B = VALUE-B
```

```

... REC RECORD-C WHERE KEY-C = VALUE-C HOLD
DB-MODIFY REF RECORD-A
... REF RECORD-B
... REC RECORD-C

```

SQL

Update specific columns of specific rows using selection criteria and alternate data from different areas for each column.

```

DB-MODIFY REC D2TAB-REC
... PM_UNIT_BASE_PRICE (25.99)
... PM_UNITS (:WS-UNITS)
... WHERE PM_PART_SHORT_DESC = 'WIDGET'
... AND PM_COLOR = 'RED'
... AND PM_UNIT_BASE_PRICE < 50

```

VSAM Batch

Hold CUST-RECORD for modification; specify an alternate storage area to contain the data that updates the record.

```

DB-OBTAIN REC CUST-RECORD
... WHERE CUST-NUMBER = SCREEN-CUST-NUMBER
IF OK-ON-REC
  DB-MODIFY REC CUST-RECORD
  ... FROM CUST-RECORD-UPDATE-AREA

```

VSAM Online

Hold CUST-RECORD for modification; specify an alternate storage area to contain the data that updates the record.

```

DB-OBTAIN REC CUST-RECORD
... WHERE CUST-NUMBER = SCREEN-CUST-NUMBER
... HOLD
IF OK-ON-REC
  DB-MODIFY REC CUST-RECORD
  ... FROM CUST-RECORD-UPDATE-AREA

```

DB-OBTAIN

Category: Database call (see *Database Calls*)

Description: Read and retrieve a database record.

Under IDMS DB, DB-OBTAIN:

- Retrieves or finds a record stored within an IDMS database
- Moves a record into Working-Storage
- Establishes currency for a given record type

Under IMS, DB-OBTAIN retrieves segments at any level and performs:

- Qualified and unqualified record retrievals
- Retrieval of records qualified on multiple fields (Boolean qualification)
- Retrieval of records qualified on secondary indices composed of more than one field
- Compound calls
- Path calls
- Hold segments for modification or deletion
- Current positioning
- Reset positioning
- First/last qualification
- Naming of an alternate I/O area that a record is read into or stored from
- Retrieval of segments using concatenated key qualification

Under SQL, DB-OBTAIN:

- Selects an entire row qualified on one column
- Selects an entire row qualified on multiple columns
- Selects specific columns from a row

- Names an alternate data name into which the host-variable structure is moved
- Joins columns from more than one table
- Specifies a union

Under VSAM, DB-OBTAIN:

- Reads records from a file
- Provides a starting point in a file that meets specified key criteria in the call
- Reads calls sequentially, in ascending key order (sequential DB-OBTAIN)
- Reads a record based upon specified key criteria (direct DB-OBTAIN)
- Positions a file for a sequential DB-OBTAIN, based upon key criteria (positional DB-OBTAIN)

Syntax: IDMS DB

Format 1, based on a CALC key or an indexed or sorted set:

```
DB-OBTAIN REC|REF recordname
... WHERE keyname = value [NEXT]
... [HOLD] [EXCLUSIVE]
```

Format 2, where the most recently retrieved occurrence of a record type is current of run unit:

```
DB-OBTAIN REC|REF recordname CURRENT
... [HOLD] [EXCLUSIVE]
```

Format 3, where the most recently retrieved occurrence of any record in the set or area is current of run unit:

```
DB-OBTAIN REC|REF IDMSREC
... SET setname|AREA areaname CURRENT
... [HOLD] [EXCLUSIVE]
```

Format 4, based on database address:

```
DB-OBTAIN REC|REF IDMSREC|recordname
... WHERE DBKEY = value
... [HOLD] [EXCLUSIVE]
```

Format 5, where the set owner is obtained when record is unknown:

```
DB-OBTAIN REC|REF IDMSREC
... SET setname OWNER
... [HOLD] [EXCLUSIVE]
```

Format 6, based on a CALC key or an indexed or sorted set, using a valid operator:

```
DB-OBTAIN [REF recordname1] REC|REF recordname2
... WHERE sortkey operator value
... [SET setname [RESET]]|[RESET]
... [HOLD] [EXCLUSIVE]
```

Format 7, based on position within set:

```
DB-OBTAIN [REF recordname1] REC|REF recordname2
... [SET setname]
... [WHERE SEQUENCE = number|FIRST|LAST|PREV|NEXT]
... [HOLD] [EXCLUSIVE]
```

Format 8, based on position within the set when the record is unknown:

```
DB-OBTAIN REC|REF IDMSREC SET setname|AREA areaname
... [WHERE SEQUENCE = number|FIRST|LAST|PREV|NEXT]
... [HOLD] [EXCLUSIVE]
```

Format 9, based on position within area:

```
DB-OBTAIN REC|REF recordname [AREA areaname]
... [WHERE SEQUENCE = number|FIRST|LAST|PREV|NEXT]
... [HOLD] [EXCLUSIVE]
```

IMS DB

Format 1, unqualified:

```
DB-OBTAIN REC recordname [HOLD] [RESET]
```

Format 2, qualified:

```
DB-OBTAIN REC recordname
... WHERE fieldname1 operator value
... [AND|OR fieldname2 operator value [FIRST|LAST]
... [INTO dataarea]
.
.
... [AND|OR fieldnameN operator value] [FIRST|LAST]
... [INTO dataarea] [HOLD]]] [RESET]
```

Format 3, qualified on secondary index values:

```
DB-OBTAIN REC recordname
... WHERE fieldname operator (value1 [ ... valueN])
... [INTO dataarea]
... [FIRST|LAST] [HOLD] [RESET]
```

Format 4, qualified compound retrieval:

```
DB-OBTAIN REF segmentname1 WHERE fieldname1 operator value
... REC|REF segmentname2 WHERE fieldname2 operator value
      .
      .
      .
... [REC segmentnameN WHERE fieldnameN operator value
... [FIRST|LAST] [HOLD]]
... [RESET]
```

Format 5, retrieve next segment:

```
DB-OBTAIN NEXT[REC] INTO dataarea
...VIEW pcbname|PCB pcbname [HOLD] [RESET]
```

Format 6, retrieve segment specified in program at run time:

```
MOVE 'segmentname' TO segmentname
DB-OBTAIN IMSREC segmentname FROM dataarea
... VIEW pcbname|PCB pcbname
```

Format 7, retrieve dependent of current record:

```
DB-OBTAIN REF recordname1 CURRENT
... REC recordname2 [WHERE fieldname operator value]
```

Format 8, retrieve segment from PSB with multiple PCBs:

```
DB-OBTAIN REC recordname
... [WHERE fieldname operator value]
... VIEW pcbname|PCB pcbname
```

Format 9, retrieve qualified or subscripted field:

```
DB-OBTAIN REC recordname
... WHERE keyname operator fieldname
... OF dataarea|SUB (number)
```

Format 1, retrieve dependent record via concatenated key:

```
DB-OBTAIN REC recordname CKEYED dataname
```

Format 1, qualified, select all columns:

```
DB-OBTAIN REC copylibname-REC
... WHERE column operator [:]altvalue|column
... [AND|OR column operator [:]altvalue|column]
.
.
... [AND|OR column operator [:]altvalue|column]
... [INTO dataname]
```

Format 2, unqualified, select all columns:

```
DB-OBTAIN REC copylibname-REC
... [INTO dataname]
```

Format 3, join all columns from two tables:

```
DB-OBTAIN REC correlname1.copylibname-REC
... REC correlnameN.copylibname-REC
... [WHERE correlname.column oper
[:]altvalue|correlname.column
... [AND|OR correlname.column op
[:]altvalue|correlname.column]
.
.
... [AND|OR correlname.column oper
[:]altvalue|correlname.col]]
```

Format 4, select specific columns:

```
DB-OBTAIN REC copylibname-REC [DISTINCT]
... column1 [(altvalue)] [... columnN [(altvalue)]]
... [WHERE [correlname.]column operator [:]altvalue|column]
```

VSAM Batch**Format 1, sequential:**

```
DB-OBTAIN REC recordname
... [VIEW keyname] [INTO dataarea] [RESET]
```

Format 2, direct:

```
DB-OBTAIN REC recordname
... WHERE keyname operator value [SUB value]
... [OF dataarea] [INTO dataarea]
```

Format 3, positional:

```
DB-OBTAIN REF recordname
... WHERE keyname operator value [SUB value] [OF dataarea]
```

VSAM Online**Format 1, sequential:**

```
DB-OBTAIN REC recordname
... [VIEW keyname] [INTO dataarea]
... [HOLD] [PREV] [REQID number] [RESET]
... [SYSID systemname] [DDN ddname]
```

Format 2, direct:

```
DB-OBTAIN REC recordname
... WHERE keyname operator value [SUB value]
... [OF dataarea] [INTO dataarea] [KLEN value]
... [HOLD] [REQID number]
... [SYSID systemname] [DDN ddname]
```

Format 3, positional:

```
DB-OBTAIN REF recordname
... WHERE keyname operator value [SUB value] [OF dataarea]
... [KLEN value] [RESETBR] [REQID number]
... [SYSID systemname] [DDN ddname]
```

Parameters: `[:](altvalue)`

Alternate value; can be a literal, column name, or host-variable, as follows.

- A host-variable is any COBOL data item referenced in your APS/SQL code; can be data item generated automatically by APS/SQL to match a DB2 column name.
- An alternate host-variable is one you instruct APS/SQL to use instead of the automatically generated one for a column.
- If you name alternate host-variables for specific columns, do not code INTO, which names an entire alternate host structure.

	<ul style="list-style-type: none"> • Precede host-variables and alternate host-variables with a colon. APS generates a # symbol for the colon.
AND <i>col op xol[:]altval</i>	Value can be a literal or data name. See also the <i>altvalue</i> parameter above.
AREA <i>areaname</i>	IDMS area name.
CKEYED <i>dataname</i>	Single data area containing concatenated key information.
<i>correlname.</i>	Correlation name (maximum 18 characters); end with a period. Required if columnname is in a select list or if the WHERE clause appears in multiple joined tables.
CURRENT	Under IDMS DB, process the last record of the type accessed. See also "Comments" below. Under IMS DB, establish positioning at the specified dependent of the current, previously-read parent.
DBKEY= <i>value</i>	Assign value to DB key.
DDN <i>ddname</i>	Specify file <i>ddname</i> ; can be a literal or data name defined as PIC X(8). Supply a value to the name option of CICS DATASET.
DISTINCT	Eliminate all but one row from each set of duplicate rows. Duplicate rows have identical selected columns from the results table.
EXCLUSIVE	IDMS keep exclusive; place an exclusive lock on the record.
FIRST	Under IDMS DB, retrieve the first record in the set. Under IMS DB, establish positioning at the first occurrence of the specified segment; generate IMS code F.

FROM <i>dataarea</i>	Alternate I/O area where program deletes, modifies, or adds a record. Required for a record obtained from an I/O area other than the default I/O area, such as by DB-OBTAIN INTO.
HOLD	Hold a record for modification or deletion. Code only once, at end of call. Do not code with PREV or KLEN. See also "Comments" below. Under IDMS DB, keep DML; place an explicit shared lock on the record.
IDMSREC	Retrieve IDMS record; do not specify a record name.
IMSREC <i>segmentname</i>	Specify that the segment name (maximum 8 characters) is in a Working-Storage variable for the program to read, modify, add, or delete. See also "Comments" below.
INTO <i>dataarea</i>	Specify I/O area where the program reads a record.
INTO <i>dataname</i>	Move host variable structure into the alternate data structure <i>dataname</i> . Can code before or after WHERE. Data moves after the actual SQL call via a MOVE statement. Generated code: <pre>IF OK-ON-REC MOVE <i>hostname</i> TO <i>dataname</i></pre>
KLEN <i>value</i> or KEYLENGTH <i>value</i>	Specify number or characters in key length; full or partial length is valid. Value can be a number or a data name defined as PIC S9(4) COMP. APS generates the CICS GENERIC option for a partial key length. Do not use with HOLD.
LAST	Under IDMS DB, retrieve last record in set. Under IMS DB, establish positioning at the last occurrence of the specified segment; generate IMS code L.

NEXT	Under IDMS DB, retrieve next record in the set (default). Under IMS DB, sequentially read forward in database.
OF <i>dataarea</i>	Qualify the I/O area moving to the value field, when more than one structure in the Data Division contains the field. Optionally code IN instead of OF.
OR <i>col op col[:]altval</i>	Value can be a literal or data name. See also the <i>altvalue</i> parameter above.
OWNER	Specify owner record.
PCB <i>pcbname</i>	Synonymous with VIEW. Specify the PCB used when the PSB contains multiple PCBs for the same database. Must be last keyword in call. See also "Comments" below.
PREV[IOUS]	Perform a reverse sequential browse starting at the last record in file. Do not code with HOLD or KLEN.
REC <i>copylib</i> -REC	Specify the 01-level name of the COBOL row layout in the DCLGEN or copybook information. Copybook library name of source data. Cannot be the same as any cursor names or DB-PROCESS-ID names. See also "Comments" below.
REC <i>recordname</i>	COBOL record or IMS segment to process.
REF <i>recordname</i>	Specify a COBOL record to reference. Under IMS, the program uses the referenced segment for navigating the database. See also "Comments" below.
REQID <i>number</i>	Unique browse identifier for performing a simultaneous browse on the same key; is a single integer (0 - 9). Assign &VS-ENDBR-CONTROL = "USER" in the APS CNTL file APVSAMIN.

RESET	Reset database or file positioning to the beginning. Code only once, at end of call. Code with PREV or KLEN, to reset file position to end. Alternate reset method: prior to retrieving under VSAM Batch or Online, set RESET-OBTAIN flag to TRUE; under IMS DB, set RESET-POSITION flag to TRUE.
RESETBR	Reset active browse on key name; generate CICS RESETBR.
SEQUENCE = <i>number</i>	Specify sequence number.
SET <i>setname</i>	Specify IDMS set name. See also "Comments" below.
SUB[SCRIPT] (<i>value</i>)	Move the subscripted field value to a specified field. Value can be a data name, literal, or, under VSAM Batch or Online, an integer.
SYSID <i>systemname</i>	Remote system name (maximum 4 characters); can be a literal region name or a Working-Storage field.
VIEW <i>keyname</i>	Specify key primary or alternate key.
VIEW <i>pcbname</i>	Synonymous with PCB. Specify the PCB used when the PSB contains multiple PCBs for the same database. Must be last keyword in call. See also "Comments" below.
WHERE <i>column op [:]altval</i>	Column is the column on which to qualify the selection. Operator can be: =, ^=, >, <, >=, <=, native SQL predicates (such as LIKE and BETWEEN). See also the <i>altvalue</i> parameter above.
WHERE <i>fld key op val fld</i>	Under VSAM Batch or Online, <i>operator</i> can be: =, EQ, >=, GTEQ; otherwise <i>operator</i> can be: =, EQ, >, GT, <, LT, >=, GE, <=, LE, <>, NE, ^=. Value can be literal, data name, or an asterisk (*). An asterisk indicates the segment record description contains the key value.

Comments: IDMS DB

- If REF is coded without REC, the action performed is a FIND.
- In Format 1:, if an indexed or sorted set has a complex key (that is, more than one field makes up the key), a Working-Storage area named setname-KEY is generated to store the key values. Before coding DB-OBTAIN, move the desired values to the appropriate fields in the record obtained.
- In Format 6, to use multiple sets, code SET *setname*. Code RESET unless currency on the set has been established.

IMS DB

- In Format 2, *fieldname1* can be a primary key, secondary key, segment sequence field, or segment search field. If it is a secondary key made up of more than one field, the data name supplied as value must be a concatenation of all fields composing the index.
- In Format 3, the number of values listed within parentheses must equal the number of source fields making up the index, and be listed in the same order. Values can be qualified or subscripted.
- In Format 4:
 - Ensure that all segments lie along a single imaginary path that runs from the root down to the dependent segment, and are in the same logical database.
 - Specify segments in hierarchical order.
 - With a compound DB-OBTAIN, code REC for the lowest level segment.
- In Format 5:
 - Do not use with IMS call codes.
 - NEXTREC generates IMS retrieval call without segment search arguments (SSAs).
 - Do not use field qualifications with NEXTREC and IMSREC.
- In Format 6:
 - Code only one segment per call. Move the correct IMS segment name into Working-Storage before issuing the call.
 - Do not use field qualifications with NEXTREC and IMSREC.

- In Format 7:
 - Code CURRENT only with REF.
 - Code CURRENT for the first level specified.
 - Do not qualify a CURRENT segment in any other way; you can qualify segments below as usual.

VSAM Online

Always follow a DB-OBTAIN HOLD with a DB-MODIFY or DB-ERASE.

Examples: IDMS DB

Retrieve records with complex key.

```
MOVE SCRL-LAST-NAME TO CUSTOMER-LAST-NAME
MOVE SCRL-FIRST-NAME TO CUSTOMER-FIRST-NAME
DB-OBTAIN REC CUSTOMER WHERE CUSTOMER-NAME-KEY = *
```

Make recent occurrence of any record current of run unit.

```
DB-OBTAIN REC IDMSREC AREA CUST-REGION
... CURRENT
```

Obtain record based on database address.

```
DB-OBTAIN REC CUSTOMER WHERE DBKEY = WS-DBKEY
```

Obtain fifth record in the set.

```
DB-OBTAIN REF CUSTOMER REF ORDER WHERE SEQUENCE = 5
```

IMS DB

Do a compound DB-OBTAIN to obtain SEGMENT-C.

```
DB-OBTAIN REF SEGMENT-A WHERE KEY-A = VALUE-A
... REF SEGMENT-B WHERE KEY-B = VALUE-B
... REC SEGMENT-C WHERE KEY-C = VALUE-C HOLD
```

Do a path call to retrieve three segments. Use REC at each level to indicate that each record is retrieved.

```
DB-OBTAIN REC SEGMENT-A WHERE KEY-A = VALUE-A
... REC SEGMENT-B WHERE KEY-B = VALUE-B
... REC SEGMENT-C HOLD
```

Do a path call to retrieve specific records (SEGMENT-A, SEGMENT-C) at certain levels. Specify that SEGMENT-B is used for qualification only and is not retrieved.

```
DB-OBTAIN REC SEGMENT-A WHERE KEY-A = VALUE-A
... REF SEGMENT-B WHERE KEY-B = VALUE-B
... REC SEGMENT-C HOLD
```

Using position, qualify database access. First, retrieve RECORD-A. Then, retrieve RECORD-B; restrict the retrieval of RECORD-B to children of the previously retrieved RECORD-A.

```
DB-OBTAIN REC RECORD-A WHERE FIELD-1 = VALUE-1
DB-OBTAIN REF RECORD-A CURRENT
... REC RECORD-B WHERE FIELD-2 = VALUE-2
```

Retrieve the last occurrence of RECORD-B under a specific RECORD-A.

```
DB-OBTAIN REF RECORD-A WHERE FIELD-1 = VALUE-1
... REC RECORD-B LAST
```

Reference the next occurrence of RECORD-A; retrieve the first occurrence of RECORD-B under RECORD-A and read it into DATAAREA-B. Also, retrieve the first occurrence of RECORD-C under RECORD-B and read it into DATAAREA-C.

```
DB-OBTAIN REF RECORD-A
... REC RECORD-B INTO DATAAREA-B
... REC RECORD-C INTO DATAAREA-C
```

Retrieve RECORD-A based on the value of the data name FIELD-X, which is found in DATAAREA-Z.

```
DB-OBTAIN REC RECORD-A
... WHERE KEY-A = FIELD-X OF DATAAREA-Z
```

Assume FIELD-X is an array as shown. First, retrieve RECORD-A based on the value of the seventh occurrence of FIELD-X. Then, obtain a dependent record by specifying its concatenated key.

```
01 FIELD-X OCCURS 10 TIMES PIC X(10).
.
.
.
DB-OBTAIN REC RECORD-A
... WHERE KEY-A = FIELD-X SUB(7)
.
.
.
```

```
DB-OBTAIN REC RECORD-C
... CKEYED WS-FIELD
```

SQL

Select the rows where PM_PART_NO equals 123; move the data to an alternate area, WS-D2MAST-RECORD in Working-Storage.

```
DB-OBTAIN REC D2TAB-REC
... WHERE PM_PART_NO = '123'
... INTO WS-D2MAST-RECORD
```

Select only one row (if duplicates exist) based on multiple selection criteria.

```
DB-OBTAIN REC D2TAB-REC DISTINCT
... WHERE PM_PART_SHORT_DESC = 'WIDGET'
... AND PM_COLOR = 'RED'
```

Select only columns PM_PART_NO and PM_COLOR, from rows of table D2MASTER based on multiple selection criteria; eliminate duplicate rows. For column PM_PART_NO, move the data to the default destination, the COBOL host-variable of the same name; for column PM_COLOR, name an alternate destination, Working-Storage field WS-COLOR.

```
DB-OBTAIN REC D2TAB-REC DISTINCT
... PM_PART_NO PM_COLOR (WS-COLOR)
... WHERE PM_PART_SHORT_DESC = 'WIDGET'
... AND PM_COLOR = 'RED'
```

VSAM Batch

Read records sequentially by the CUST-NUMBER key.

```
DB-OBTAIN REC CUST-RECORD VIEW CUST-NUMBER
```

Read records sequentially; store record in an alternate storage area.

```
DB-OBTAIN REC CUST-RECORD
... INTO CUST-RECORD-SAVE-AREA
```

Read records by CUST-NUMBER key; identify the subscripted SCREEN-CUST-NUMBER that is used as the key search value. After successful execution, establish file position so that a sequential DB-OBTAIN can read the next record.

```
DB-OBTAIN REC CUST-RECORD
... WHERE CUST-NUMBER = SCREEN-CUST-NUMBER
```

```
... SUB (ROW-CTR)
```

Verify the existence of a CUST-NUMBER and (if successful) provide a starting point in the file for a sequential DB-OBTAIN to execute.

```
DB-OBTAIN REF CUST-RECORD
... WHERE CUST-NUMBER = SCREEN-CUST-NUMBER
IF OK-ON-REC
    DB-OBTAIN REC CUST-RECORD
    ... VIEW CUST-NUMBER
```

VSAM Online

Sequentially read records into an alternate storage area and hold a record for updating. The VSAM Generator ends the sequential read and rereads the file via the primary key for updating. The sequential read is then resumed on the next execution of the DB-OBTAIN call.

```
DB-OBTAIN REC CUST-RECORD
... INTO CUST-RECORD-SAVE-AREA
... HOLD
    .
    .
DB-MODIFY REC CUST-REC
```

Read records sequentially by the CUST-NUMBER key.

```
DB-OBTAIN REC CUST-RECORD VIEW CUST-NUMBER
```

Read records by CUST-NUMBER key. Identify the subscripted SCREEN-CUST-NUMBER that is used as the key search value. Establish file position for a sequential DB-OBTAIN.

```
DB-OBTAIN REC CUST-RECORD
... WHERE CUST-NUMBER = SCREEN-CUST-NUMBER SUB (ROW-CTR)
```

Read by ORDR-NUMBER; hold the record for updating. Note that if ORDR-NUMBER was an alternate key, the file would be reread via the primary key for updating.

```
DB-OBTAIN REC ORDR-RECORD
... WHERE ORDR-NUMBER = CUST-ORDR-NUMBER HOLD
```

Verify the existence of CUST-RECORD with the value specified in CUST-NUMBER and (if successful) provide a starting point in the file for a sequential DB-OBTAIN.

```
DB-OBTAIN REF CUST-RECORD
... WHERE CUST-NAME = SCREEN-CUST-NAME
```

```
IF OK-ON-REC
  DB-OBTAIN REC CUST-RECORD... VIEW CUST-NAME
```

DB-OPEN

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB, SQL, and VSAM batch targets

Description: Open a record, cursor set, or file.

Syntax: **IDMS DB**

```
DB-OPEN [ALL] [MODE usagemode]
... [AREA areaname]
```

SQL

```
DB-OPEN CUR[SOR] cursorname
```

OS4 and VSAM Batch

Format 1:

```
DB-OPEN FILE filename [... filename]
... MODE option
```

Format 2:

```
DB-OPEN FILE ALL
... MODE option
```

Parameters:	ALL	Specify all files and ready all areas defined in the subschema.
	AREA <i>areaname</i>	IDMS area name.
	CUR[SOR] <i>cursorname</i>	Specify cursor. Cursorname must be previously named by DB-DECLARE or DB-PROCESS-ID.
	FILE <i>filename</i>	File(s) to process.

<i>MODE option</i>	File processing mode: INPUT, OUTPUT, I-O, EXTEND
<i>MODE usagemode</i>	Usage mode--RETRIEVAL (default),PROTECTED RETRIEVAL, EXCLUSIVE RETRIEVAL, UPDATE, PROTECTED UPDATE, EXCLUSIVE UPDATE

Comments:**IDMS DB**

- DB-OPEN generates an IDMS READY command; if not coded, the call is automatically generated.
- *Usagemode*, if specified in the subschema, replaces RETRIEVAL as the default.

SQL

A cursor set can be opened, processed, and closed multiple times in the same program. A DB-OPEN issued for an open cursor set causes an error; first code a DB-CLOSE.

SAM Batch

DB-OPEN is required.

Examples: Open the DORDER file for write-only processing.

```
DB-OPEN FILE DORDER MODE EXTEND
```

IDMS DB

Ready all areas in the data view in the default usage mode.

```
DB-OPEN ALL
```

Ready CUSTOMER-REGION in PROTECTED UPDATE usage mode.

```
DB-OPEN MODE PROTECTED UPDATE AREA CUSTOMER-REGION
```

SQL

Open cursor set D2MAST-CURSOR, previously defined in DB-DECLARE.

```
DB-OPEN CURSOR D2MAST-CURSOR
```

VSAM Batch

Open the AORDER and ASUPLIR files for I-O processing.

```
DB-OPEN FILE AORDER ASUPLIR MODE I-O
```

DB-PROCESS

Category: Database call (see *Database Calls*)

Description: Combine record retrieval and looping functions into one call.

Under IDMS DB, DB-PROCESS processes all records within a set, an area, or an indexed set based on a specified value.

Under IMS, DB-PROCESS processes:

- Records that satisfy the key qualification (key qualified)
- Records sequentially (unqualified)

Under SQL, DB-PROCESS simplifies cursor row processing by:

- Declaring a cursor
- Opening a cursor for processing
- Defining a loop flag and a loop counter
- Providing logic for retrieving rows from the results table
- Executing user-written row processing code
- Processing closing the cursor set file at the end of processing
- Processing entire rows throughout a cursor set
- Processing specific columns throughout a cursor set
- Processing specific columns and rows throughout a cursor set
- Processing columns from more than one table, such as a join)

Under VSAM, DB-PROCESS processes records:

- Sequentially, beginning at the position established by the key qualification (key qualified)
- Sequentially, beginning at the beginning, end, or a previously defined position in the file (unqualified)

Syntax: IDMS DB

```
DB-PROCESS [REF recordname] REC recordname
... [DB-PROCESS-ID name] [WHERE keyname operator value]
... [SET setname|AREA areaname] [RESET] [HOLD] [EXCLUSIVE]
Controlled logic block
```

IMS DB**Format 1, key-qualified:**

```
DB-PROCESS REC recordname
... [WHERE keyname operator value [SUB value] [OF dataarea]]
... [DB-PROCESS-ID name] [INTO dataarea]
... [HOLD] [RESET]
... [VIEW pcbname|PCB pcbname]
Controlled logic block
```

Format 2, unqualified:

```
DB-PROCESS REC recordname
... [DB-PROCESS-ID name] [INTO dataarea]
... [HOLD] [RESET]
... [VIEW pcbname|PCB pcbname]
Controlled logic block
```

SQL**Format 1, unqualified--select all columns:**

```
DB-PROCESS REC copylibname-REC
... [DB-PROCESS-ID name]
... [DB-LOOP-MAX=number]
... [FETCH ONLY] [WITH HOLD]
... [OPTIMIZE number]
... [UPDATE|ORDER]
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]
... [INTO dataname]
Controlled logic block
```

Format 2, qualified--select all columns:

```
DB-PROCESS REC copylibname-REC
... [DB-PROCESS-ID name]
... [FETCH ONLY] [WITH HOLD] [OPTIMIZE number]
... WHERE column operator [[:]]altvalue|column
... [AND|OR column operator [[:]]altvalue|column]
. . .
... [AND|OR column operator [[:]]altvalue|column]
```

```

... [DB-LOOP-MAX=number]
... [UPDATE|ORDER
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]
... [INTO dataname]
Controlled logic block

```

Format 3, select specific columns:

```

DB-PROCESS REC copylibname-REC
... [DB-PROCESS-ID name]
... [DISTINCT]
... [FETCH ONLY]
... [WITH HOLD]
... [OPTIMIZE number]
... column1 [(altvalue)] [... columnN [(altvalue)]]
... WHERE column operator [[:]altvalue]|column
... [AND|OR column operator [[:]altvalue]|column]
.
.
.
... [AND|OR column operator [[:]altvalue]|column]
... [DB-LOOP-MAX=number]
... [UPDATE|ORDER
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]
Controlled logic block

```

Format 4, join columns from two or more tables:

```

DB-PROCESS REC correlname.copylibname-REC
... [DB-PROCESS-ID name] [DISTINCT]
... [FETCH ONLY] [WITH HOLD]
... [OPTIMIZE number]
... [column1 [(altvalue)] [... columnN [(altvalue)]]]
... REC correlname.copylibname-REC
... [column1 [(altvalue)] [... columnN [(altvalue)]]]
.
.
... [WHERE correlname.column oper
[[:]altvalue]|correlname.column
... [AND|OR correlname.column oper
[[:]altvalue]|correlname.col]
.
.
... [AND|OR correlname.column oper
[[:]altvalue]|correlname.col]
... [DB-LOOP-MAX=number]
... [ORDER
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]
Controlled logic block

```

Format 5, specify a UNION:

```

DB-PROCESS REC copylibname-REC
... [DB-PROCESS-ID name]
... [DISTINCT]
... [FETCH ONLY]
... [WITH HOLD]
... [OPTIMIZE number]
... [column1 [(altvalue)] [... columnN [(altvalue)]]]
... [WHERE column operator [[:]altvalue]|column]
... [AND|OR column operator [[:]altvalue]|column]
.
.
.
... [AND|OR column operator [[:]altvalue]|column]
... [DB-LOOP-MAX=number]
... UNION [ALL]
DB-OBTAIN REC copylibname-REC
.
.
.
... [ORDER]
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]
Controlled logic block

```

VSAM Batch**Format 1, qualified:**

```

DB-PROCESS REC recordname WHERE keyname operator value
... [DB-PROCESS-ID name] [SUB value]
... [OF dataarea] [INTO dataarea]
Controlled logic block

```

Format 2, unqualified:

```

DB-PROCESS REC recordname [VIEW keyname]
... [DB-PROCESS-ID name] [INTO dataarea] [RESET]
Controlled logic block

```

VSAM Online**Format 1, key-qualified:**

```

DB-PROCESS REC recordname WHERE keyname operator value
... [DB-PROCESS-ID name] [SUB value]
... [OF dataarea] [INTO dataarea]
... [KLEN value] [HOLD] [PREV]
... [REQID number] [SYSID systemname]

```

```
... [DDN ddname]
  Controlled logic block
```

Format 2, unqualified:

```
DB-PROCESS REC recordname
... [DB-PROCESS-ID name] [INTO dataarea]
... [HOLD] [PREV] [RESET]
... [REQID number] [SYSID systemname]
... [VIEW keyname] [DDN ddname]
  Controlled logic block
```

Parameters: [:(*altvalue*)

Alternate value; can be a literal, column name, or host-variable, as follows.

- A host-variable is any COBOL data item referenced in your APS/SQL code; can be data item generated automatically by APS/SQL to match a DB2 column name.
- An alternate host-variable is one you instruct APS/SQL to use instead of the automatically generated one for a column.
- If you name alternate host-variables for specific columns, do not code INTO, which names an entire alternate host structure.
- Precede host-variables and alternate host-variables with a colon. APS generates a # symbol for the colon.

AND *col op col*[:]*altval*

Value can be a literal or data name. See also the *altvalue* parameter above.

AREA *areaname*

IDMS area name.

BROWSE *col1 colN*

Modify columns during cursor processing. In cursor processing, you cannot modify a column unless you specify UPDATE first. Do not code BROWSE with ORDER, UNION, DISTINCT, GROUP BY, or if the call specifies a join or selects column functions.

CKEYED	Retrieve segment with a concatenated key.
<i>correlname.</i>	Correlation name (maximum 18 characters); end with a period. Required if <i>columnname</i> is in a select list or if the WHERE clause appears in multiple joined tables.
CURRENT	Under IDMS DB, process the last record of the type accessed. Under IMS DB, establish positioning at the specified dependent of the current, previously-read parent.
DB-LOOP-MAX= <i>number</i>	Maximum number loops allowed. Overrides loop flags for current structure only; you can define a different limit for each DB-PROCESS. See also "Comments" below.
DB-PROCESS-ID <i>name</i>	Generate: <ul style="list-style-type: none"> • Cursor named <i>name</i> (for SQL targets) • End-process flag named <i>name</i>-END-PROCESS • Counter named <i>name</i>-PROCESS-CTR <i>Name</i> (maximum 12 characters) must be unique, and cannot be the same as the subschema copylib names. See also "Comments" below.
DDN <i>ddname</i>	Specify file <i>ddname</i> ; can be a literal or data name defined as PIC X(8). Supply a value to the name option of CICS DATASET.
DISTINCT	Eliminate all but one row from each set of duplicate rows. Duplicate rows have identical selected columns from the results table.
EXCLUSIVE	IDMS keep exclusive; place an exclusive lock on the record.

FETCH ONLY	Specify that the table is read-only and therefore the cursor cannot be referred to in positioned UPDATE and DELETE statements. Do not code in a call that contains an UPDATE clause.
FIRST	Under IDMS DB, retrieve the first record in the set. Under IMS DB, establish positioning at the first occurrence of the specified segment; generate IMS code F.
HOLD	Hold a record for modification or deletion. Code only once, at end of call. Do not code with PREV or KLEN. Under IDMS DB, keep DML; place an explicit shared lock on the record.
IMSREC <i>segmentname</i>	Specify that the segment name (maximum 8 characters) is in a Working-Storage variable for the program to read, modify, add, or delete.
INTO <i>dataarea</i>	Specify I/O area where the program reads a record.
INTO <i>dataname</i>	Move host variable structure into the alternate data structure <i>dataname</i> . Can code before or after WHERE. Data moves after the actual SQL call via a MOVE statement. Generated code: <pre>IF OK-ON-REC MOVE <i>hostname</i> TO <i>dataname</i></pre>
KLEN <i>value</i> or KEYLENGTH <i>value</i>	Specify number or characters in key length; full or partial length is valid. <i>Value</i> can be a number or a data name defined as PIC S9(4) COMP. APS generates the CICS GENERIC option for a partial key length. Do not use with HOLD or PREV.
LAST	Under IDMS DB, retrieve last record in set. Under IMS DB, establish positioning at the last occurrence of the specified segment; generate IMS code L.

NEXT	Under IDMS DB, retrieve next record in the set (default). Under IMS DB, sequentially read forward in database.
OF <i>dataarea</i>	Qualify the I/O area moving to the value field, when more than one structure in the Data Division contains the field. Optionally code IN instead of OF.
OPTIMIZE <i>number</i>	Specify estimated maximum number of rows that call will retrieve. If the call retrieves no more than <i>number</i> rows, performance could be improved. Specifying this keyword does not prevent all rows from being retrieved.
OR <i>col op col[:altval]</i>	<i>Value</i> can be a literal or data name. See also the <i>altvalue</i> parameter above.
ORDER [ASC DESC] [<i>col1 colM</i>]	Sort the results table in ascending (default) or descending order, based on the values in the columns specified. Specify the column either by name or by relative position in the column selection list. Specify at least one column. Do not code with UPDATE.
PCB <i>pcbname</i>	Synonymous with VIEW. Specify the PCB used when the PSB contains multiple PCBs for the same database.
PREV[IOUS]	Perform a reverse sequential browse starting at the last record in file. Do not code with HOLD or KLEN.
REC <i>copylib-REC</i>	Specify the 01-level name of the COBOL row layout in the DCLGEN or copybook information. Cannot be the same as any cursor names or DB-PROCESS-ID names.
REC <i>recordname</i>	COBOL record or IMS segment to process.
REF <i>recordname</i>	Specify a COBOL record to reference. Under IMS, the program uses the referenced segment for navigating the database.

REQID <i>number</i>	Unique browse identifier for performing a simultaneous browse on the same key; is a single integer (0 - 9). Assign &VS-ENDBR-CONTROL = "USER" in the APS CNTL file APVSAMIN.
RESET	Reset database or file positioning to the beginning. Code only once per call, at end. Code with PREV or KLEN, to reset file position to end. Alternate reset method: prior to retrieving under VSAM Batch or Online, set RESET-OBTAIN flag to TRUE; under IMS DB, set RESET-POSITION flag to TRUE.
SUB[SCRIPT] (<i>value</i>)	Move the subscripted field value to a specified field. Value can be a data name, literal, or, under VSAM Batch or Online, an integer.
SYSID <i>systemname</i>	Remote system name (maximum 4 characters); can be a literal region name or a Working-Storage field.
UPDATE <i>col1 colN</i>	Modify columns during cursor processing. In cursor processing, you cannot modify a column unless you specify UPDATE first. Do not code UPDATE with ORDER, UNION, DISTINCT, GROUP BY, or if the call specifies a join or selects column functions.
VIEW <i>keyname</i>	Specify primary or alternate key.
VIEW <i>pcbname</i>	Synonymous with PCB. Specify the PCB used when the PSB contains multiple PCBs for the same database.
WHERE <i>col op [:]altval</i>	<i>Column</i> is the column on which to qualify the selection. <i>Operator</i> can be: =, ^=, >, <, >=, <=, native SQL predicates (such as LIKE and BETWEEN). See also the <i>altvalue</i> parameter above.

<i>WHERE fld key op value</i>	<p>Under VSAM Batch or Online, <i>operator</i> can be: =, EQ, >=, GTEQ; otherwise <i>operator</i> can be: =, EQ, >, GT, <, LT, >=, GE, <=, LE, <>, NE, ^=.</p> <p>Value can be literal, data name, or an asterisk (*). An asterisk indicates the segment record description contains the key value.</p>
WITH HOLD	Prevent the closing of a cursor as a consequence of a commit operation. See also "Comments" below.

Loop Structure

DB-PROCESS provides a built-in loop structure to process records and rows. The loop structure can include blocks of user-supplied logic and APS-generated control fields. User logic executes once for each successful iteration of DB-PROCESS, and can include

- Any APS/DB call, except DB-OPEN, DB-DECLARE, or DB-CLOSE
- Record and row processing code, for example, MOVE DATA TO SCREEN

APS generates the following fields, enabling you to logically terminate the loop structure.

APS-END-PROCESS	S-COBOL flag initialized FALSE. To end the process loop, set flag to TRUE.
<i>name</i> -END-PROCESS	APS generates this flag, where <i>name</i> is the PROCESS-ID name. Use this flag when using DB-PROCESS-ID clauses for nested loops.
<i>name</i> -PROCESS-CTR	APS generates this counter, where <i>name</i> is the PROCESS-ID name. Use this counter when using DB-PROCESS-ID clauses for nested loops.
APS-PROCESS-CTR	<p>APS increments this counter at each process loop execution. This counter</p> <ul style="list-style-type: none"> • Controls looping. • Serves as a subscript when moving data into a table or screen fields. • Counts processed records.

IMS DB, VSAM Batch, and VSAM Online

Sample loop syntax:

```

/* Begin process loop
DB-PROCESS REC recordname
... WHERE keyname operator value parameters
... [DB-PROCESS-ID name]
  IF APS-PROCESS-CTR|name-PROCESS-CTR > value
  /* End process loop
      TRUE APS-END-PROCESS|name-END-PROCESS
  ELSE
      /* User-written record processing logic
IF ...
/*Logic executed after process loop termination
/* Includes file status checking

```

IDMS DB

Sample loop syntax:

```

DB-OBTAIN REC recordname WHERE key = value
/* Begin process loop
DB-PROCESS REF recordname REC recordname
  IF APS-PROCESS-CTR > value
  /* End process loop
      TRUE APS-END-PROCESS
  ELSE
      /* User-written record processing logic
      /* Can include file status checking
/* Logic executed after process loop termination
/* Can include file status checking

```

SQL

Sample loop syntax:

```

/* Begin process loop
DB-PROCESS REC recordname
  /* Custom row processing code
  IF APS-PROCESS-CTR|name-PROCESS-CTR > value
  /* End process loop
      TRUE APS-END-PROCESS|name-END-PROCESS
  ELSE
      /* Custom record processing logic
      /* Can include file status checking
/* Logic executed after process loop termination
/* Can include file status checking

```

- Comments:**
- Under IDMS DB and SQL, code file status checking inside a DB-PROCESS loop; under IMS DB, and VSAM Batch or Online, code file status checking outside the process loop.
 - You can nest DB-PROCESS calls if you supply each call with a unique DB-PROCESS-ID clause, thus providing controlled exits for each.
 - DB-PROCESS continues looping until the program reaches a TRUE APS-END-PROCESS or TRUE name-END-PROCESS, an invalid or end of file condition, the limit defined by an APS internal counter.
 - You can change the value of LOOP-MAX in the APSMACS file; this affects all generated applications. Or, change the LOOP-MAX limit at the program level by redefining it at the top of your program with the SYM1 or SYM2 keyword. For example:

```
SYM1      % &prefix-LOOP-MAX = 200
```

- APS defines a loop limit as follows:
 - If you do not code *DB-PROCESS-ID*, the loop limit is 500, set by the APS counter &APS-DB-PROCESS-GLOBAL-LIMIT. This counter resides in the APSMACS file APSBASE.
 - If you do code *DB-PROCESS-ID*, adhere to the following loop limit and counter for each target.

Target	Limit	APS Counter
IDMS	100	&IDMS-LOOP-MAX
IMS DB	100	&VS-IMS-LOOP-MAX
SQL	100	&DB2-LOOP-MAX
VSAM Batch	999,999	&VS-MVS-LOOP-MAX
VSAM Online	100	&VS-CICS-LOOP-MAX

IMS DB

Processing begins at a previously established position in the database or, if RESET coded, at the beginning of the database.

SQL

- DB-PROCESS generates and performs the DB-DECLARE, DB-OPEN, DB-FETCH, and DB-CLOSE calls within loop structure.

- APS automatically declares the cursor at the first DB-PROCESS. All subsequent calls within the loop reference the cursor. The cursor is named
 - By you, if you code DB-PROCESS-ID.
 - By APS, which generates a copylib name and a numeric suffix, beginning with one and increasing by one with each subsequent DB-PROCESS for the same DCLGEN or copybook member.
- When you code WITH HOLD, a commit operation commits all the changes in the current unit of work, but releases only locks that are not required to maintain the cursor. Afterwards, you must code an initial DB-FETCH before you can execute a positioned update or delete. After the initial DB-FETCH, the cursor is positioned on the row following the one it was positioned on before the commit operation.
- The WITH HOLD clause is ignored in CICS and IMS DC.

VSAM Batch

- The ENDFILE condition can be determined outside of the process loop, even after DB-PROCESS ends the sequential browse.
- You must precede a DB-PROCESS call with a DB-OPEN call, and code a subsequent DB-CLOSE call. APS does not support automatic file opening and closing.

VSAM Online

- Because DB-PROCESS terminates the active browse when the process loop is terminated, it is not necessary to end the browse with the DB-FREE call.
- DB-PROCESS unlocks a locked record when the process loop terminates. Unlocking a record via a DB-FREE UNLOCK is unnecessary.
- The ENDFILE condition can be determined outside of the process loop, even after DB-PROCESS ends the sequential browse.

Examples: IMS DB, VSAM Batch, and VSAM Online

Process CUST-RECORD; execute logic upon each successful read. Note that CUST-PROCESS-CTR serves as a subscript for the screen fields.

```

DB-PROCESS REC CUST-RECORD
... WHERE CUST-KEY >= SCREEN-KEY
... DB-PROCESS-ID CUST
      IF CUST-PROCESS-CTR > SCREEN-MAX
        TRUE CUST-END-PROCESS
      ELSE
        SCREEN-CUST      (CUST-PROCESS-CTR) = CUST-NAME
        SCREEN-PHONE    (CUST-PROCESS-CTR) = CUST-PHONE
        SCREEN-ADDRESS  (CUST-PROCESS-CTR) = CUST-ADDRESS
IF NTF-ON-REC
  SCREEN-MSG = 'CUSTOMER NOT FOUND'
ELSE-IF END-ON-REC
  SCREEN-MSG = 'END OF CUSTOMER RECORDS'

```

IDMS DB

Process CUST-RECORD; execute logic upon each successful read.

```

DB-PROCESS REC CUST-RECORD
... WHERE CUST-KEY >= SCREEN-KEY
... SET CUST-ORDR
... DB-PROCESS-ID CUST
      IF CUST-PROCESS-CTR > SCREEN-MAX
        TRUE CUST-END-PROCESS
      ELSE
        SCREEN-CUST      (CUST-PROCESS-CTR) = CUST-NAME
        SCREEN-PHONE    (CUST-PROCESS-CTR) = CUST-PHONE
        SCREEN-ADDRESS  (CUST-PROCESS-CTR) = CUST-ADDRESS
IF NTF-ON-REC
  SCREEN-MSG = 'CUSTOMER NOT FOUND'
ELSE-IF END-ON-REC
  SCREEN-MSG = 'END OF CUSTOMER RECORDS'

```

SQL

Declare, name, and open a cursor; retrieve all rows and columns; process only one row from duplicate rows; close the cursor; move data into an alternate area.

```

DB-PROCESS REC D2TAB-REC
... DB-PROCESS-ID D2MAST-ID
... DISTINCT
... INTO WS-D2MAST-RECORD

```

Select specific columns from selected rows; name alternate host-variables for specific columns; sort columns by position in selection list.

```
DB-PROCESS REC D2TAB-REC
... DB-PROCESS-ID D2MAST-ID
... PM_PART_NO (WS-PART-NO)
... PM_NEW_PART_NO (WS-NEW-PART-NO) PM_UNITS
... ORDER 1 ASC 3 DESC
```

Nest one DB-PROCESS within another.

```
DB-PROCESS REC A.D2TAB-REC
... DB-PROCESS-ID D2MAST-ID
... DISTINCT
... PM_PART_NO PM_UNITS PM_PART_SHORT_DESC
  IF D2MAST-ID-PROCESS-CTR >5
    TRUE D2MAST-ID-END-PROCESS
  ELSE
    /* Custom logic to process SAVEKEY
    IF NOT TRUE D2MAST-ID-END-PROCESS
      DB-PROCESS REC D2INV-REC
      ... DB-PROCESS-ID D2INV-ID
      ... WHERE INPM_PART_NO = :WS-SAVE-PART-NO
      /* Custom record processing logic
```

DB-ROLLBACK

Category: Database call (see *Database Calls*)

Compatibility: SQL target

Description: Perform SQL ROLLBACK functions.

Syntax: DB-ROLLBACK

DB-STORE

Category: Database call (see *Database Calls*)

Description: Write a record to a file or database.

Under IMS, DB-STORE lets you:

- Specify either a single record level or multiple records along a path, like DB-OBTAIN.
- Store a dependent segment--one that references one or more parent level segments to specify the exact record placement. This is a compound DB-STORE.

Under SQL, DB-STORE lets you add entire rows or selected columns.

Syntax: **IDMS DB**

```
DB-STORE REC recordname
```

IMS DB

```
DB-STORE [REC|REF recordname1] [FROM dataarea]
... [VIEW pcbname|PCB pcbname]
... [WHERE fieldname operator value]
... [REC|REF recordname2] [FROM dataarea]
... [VIEW pcbname|PCB pcbname]
... [WHERE fieldname operator value]
... [SUB number] [OF dataarea]
.
.
.
... REC recordnameN [FROM dataarea]
... [VIEW pcbname|PCB pcbname]
... [WHERE fieldname operator value]
... [SUB number] [OF dataarea]
```

OS4

```
DB-STORE REC recordname
... [FORMAT formatname]
... [FROM dataarea]
```

SQL

Format 1:

```
DB-STORE REC copylibname-REC
... [column1 [(altvalue)] [... columnN [(altvalue)]]]
... [FROM dataname]
```

Format 2:

```
DB-STORE REC copylibname-REC
... [column1 [(altvalue)] [... columnN [(altvalue)]]]
... [DB-OBTAIN REC copylibname-REC
... column1 [... columnN]
... WHERE column1 operator [:]altvalue
... [AND|OR column2 operator [:]altvalue]]
...
... [AND|OR columnN operator [:]altvalue ]
```

VSAM Batch

```
DB-STORE REC recordname [FROM dataarea]
```

VSAM Online

```
DB-STORE REC recordname [FROM dataarea]
... [SYSID systemname] [DDN ddname]
```

Parameters: [:](*altvalue*)

Alternate value; can be a literal, column name, or host-variable, as follows.

- A host-variable is any COBOL data item referenced in your APS/SQL code; can be a data item generated automatically by APS/SQL to match a DB2 column name.
- An alternate host-variable is one you instruct APS/SQL to use instead of the automatically generated one for a column.
- If you name alternate host-variables for specific columns, do not code FROM, which names an entire alternate host structure.

	<ul style="list-style-type: none"> • Precede host-variables and alternate host-variables with a colon. APS generates a # symbol for the colon.
AND <i>col op [:]altval</i>	Value can be a literal or data name. See also the <i>altvalue</i> parameter above.
CKEYED	Position to correct parent segments.
DDN <i>ddname</i>	Specify file <i>ddname</i> ; can be a literal or data name defined as PIC X(8). Supply a value to the name option of CICS DATASET.
FROM <i>dataarea</i>	Alternate I/O area where program deletes, modifies, or adds a record. Required for a record obtained from an I/O area other than the default I/O area, such as by DB-OBTAIN INTO. See also "Comments" below.
FROM <i>dataname</i>	Move alternate data structure to the host variable structure name. Data moves prior to the actual SQL call via MOVE statement. Preferred format is to code FROM before WHERE, otherwise you must separate the WHERE and FROM with ENDWHERE. See also "Comments" below.
IMSREC <i>segmentname</i>	Specify that the segment name (maximum 8 characters) is in a Working-Storage variable for the program to read, modify, add, or delete;.
OF <i>dataarea</i>	Qualify the I/O area moving to the value field, when more than one structure in the Data Division contains the field. Optionally code IN instead of OF.
OR <i>col op [:]altval</i>	Value can be a literal or data name. See also the <i>altvalue</i> parameter above.
PCB <i>pcbname</i>	Synonymous with VIEW. Specify the PCB used when the PSB contains multiple PCBs for the same database. See also "Comments" below.

REC <i>copylib-REC</i>	Specify the 01-level name of the COBOL row layout in the DCLGEN or copybook information. Cannot be the same as any cursor names or DB-PROCESS-ID names.
REC <i>recordname</i>	COBOL record or IMS segment to process.
REF <i>recordname</i>	Specify a COBOL record to reference. Under IMS, the program uses the referenced segment for navigating the database.
SUB[SCRIPT](<i>value</i>)	Move the subscripted field value to a specified field. Value can be a data name, literal, or, under VSAM Batch or Online, an integer.
SYSID <i>systemname</i>	Remote system name (maximum 4 characters); can be a literal region name or a Working-Storage field.
VIEW <i>pcbname</i>	Synonymous with PCB. Specify the PCB used when the PSB contains multiple PCBs for the same database. See also "Comments" below.
WHERE <i>col op [:]altval</i>	Column is the column on which to qualify the selection. Operator can be: =, ^=, >, <, >=, <=, native SQL predicates (such as LIKE and BETWEEN). See also the <i>altvalue</i> parameter above.
WHERE <i>fld key oper value</i>	Code only with REF parameter. Under VSAM Batch or Online, <i>operator</i> can be: =, EQ, >=, GTEQ; otherwise <i>operator</i> can be: =, EQ, >, GT, <, LT, >=, GE, <=, LE, <>, NE, ^=. Value can be literal, data name, or an asterisk (*). An asterisk indicates the segment record description contains the key value.

Comments: IDMS DB

Initialize all CALC keys; make all sets, in which the object record participates as a mandatory member, current of record type. If the location mode of the object record is VIA, establish currency for the set in which the record participates as a member.

IMS DB

- When a record is not obtained from the default I/O area, such as by DB-OBTAIN INTO, code FROM *dataarea*.
- When a record is not obtained from the default PCB, such as by DB-OBTAIN VIEW|PCB, code VIEW|PCB *pcbname*.
- When you store a new root, DB-STORE places the record according to its key field value. If a STORE for a dependent record provides qualification at each parent level, this qualification specifies the position of the new record. Otherwise, prior database positioning determines record placement. Thus, a DB-STORE for a dependent record should either
 - Specify qualification at each parent level, or
 - Ensure that the preceding database call executed for the database PCB accesses the desired parent.

Examples: IDMS DB

Write record type ORDER from Working-Storage.

```
DB-STORE REC ORDER
```

IMS DB

Add an occurrence of RECORD-A to the database.

```
DB-STORE REC RECORD-A
```

Add one occurrence of RECORD-A and one of its dependents, RECORD-B.

```
DB-STORE REC RECORD-A REC RECORD-B
```

Add one occurrence of RECORD-B, a dependent of a specific RECORD-A.

```
DB-STORE REF RECORD-A WHERE FIELD-1 = VALUE-1 REC RECORD-B
```

Alternately, add RECORD-B by coding DB-STORE after DB-OBTAIN.

```
DB-OBTAIN REF RECORD-A WHERE KEY-A = VALUE-A
DB-STORE RECORD-B
```

SQL

Insert specific columns into D2MASTER; for PM_PART_NO, store information from the default COBOL host variable; for other columns, name alternate sources of information.

```
DB-STORE REC D2TAB-REC
... PM_PART_NO PM_NEW_PART_NO ('23432')
... PM_COLOR (:WS-NEW-COLOR)
```

Insert columns IN_PART_NO and IN_UNIT_BASE_PRICE from D2MASTER into table D2INVEN; select only rows from D2MASTER where PM_PART_NO does not equal PM_NEW_PART_NO.

```
DB-STORE REC D2INVEN-REC IN_PART_NO IN_UNIT_BASE_PRICE
... DB_OBTAIN REC D2MASTER-REC PM_PART_NO PM_UNIT_BASE_PRICE
... WHERE PM_PART_NO not= PM_NEW_PART_NO
```

VSAM Batch and VSAM Online

Write ORDR-RECORD to the file; check file status.

```
DB-STORE REC ORDR-RECORD
IF OK-ON-REC
    SCREEN-MSG = 'ORDER ADDED TO FILE'
ELSE-IF IVD-ON-REC
    SCREEN-MSG = 'ORDER ALREADY EXISTS'
```

DB-SUBSCHEMA

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB, IMS DB, SQL, VSAM batch, and VSAM online targets

Description: Include the imported program subschema in your program. APS automatically does this for you when you specify your subschema in the Application Painter after importing the subschema.

Syntax: DB-SUBSCHEMA *subschema*name

Comment: Specify a subschema for an APS program in either of the following ways.

- Recommended way Code the subschema name in the Application Painter.
- Code a DB-SUBSCHEMA call prior to the NTRY parameter.

DDIFILE Report (DB01)

Category: APS-generated report (see *Database Calls*)

Description: The DDIFILE Report describes the contents of the DDIFILE that contains your imported database definitions. Use this report to understand and evaluate the results of an import.

The report provides a section for each of the following items.

- DBDs
- PSBs
- Files
- Subschemas

Comment: Produce the DDIFILE Report from the Documentation Facility.

Example:

```
REPORT CODE: DB01          APS DATA DESCRIPTION INTERFACE          PAGE      1
                           DBD REPORT                               92/01/21  09:53
```

NAME	CREATED	UPDATED	REC NO	MESSAGE
BE3ORDER	88/08/19	88/08/19	20	INCOMPLETE SET OF DDI NAMES FOUND
BE3ORDRX	88/08/19	88/08/19	3	NO DDI NAMES SPECIFIED
CORDPSB	89/10/12	00/00/00	3	
CPTORDLD	89/11/02	00/00/00	3	

```
REPORT CODE: DB01          APS DATA DESCRIPTION INTERFACE          PAGE      4
                           FILE REPORT                               92/01/21  09:53
```

NAME	CREATED	UPDATED	REC NO	MESSAGE
CUSTMAST-REC	90/08/13	90/08/15	5	
CUSTORDR-REC	90/08/13	90/08/15	5	

REPORT CODE: DB01

APS DATA DESCRIPTION INTERFACE
SUBSCHEMA REPORTPAGE 5
92/01/21 09:53

NAME	CREATED	UPDATED	REC NO	MESSAGE
DMVINQ	90/05/10	90/12/07	1	
DMVLST	90/05/10	90/12/07	3	

```

*****      FINAL TOTALS      *****
TOTAL NUMBER OF DBDS              18
TOTAL NUMBER OF DBD RECS          530
TOTAL NUMBER OF PSBS              45
TOTAL NUMBER OF PSB RECS          93
TOTAL NUMBER OF FILES              8
TOTAL NUMBER OF FILE RECS         30
TOTAL NUMBER OF SUBSCHS           10
TOTAL NUMBER OF SUBSCH RECS       21
TOTAL NUMBER OF ENTITIES           63
TOTAL NUMBER OF RECORDS           623

```

DDI Statements

Compatibility: IMS and VSAM targets

Category: Importer feature

Description: Identify IMS database segments and VSAM files and their corresponding COBOL Syntax

DDI statements are comprised of the literal *DDI followed by a statement type and its applicable parameter parameters. The statement type identifies what database or file component to import. For example, database, record or field. Code DDI statements for any environment as follows:

Column	Value
7-10	*DDI
11	Blank
12-14	Statement type
15	Blank
16-18	Keyword parameter

IMS DB

The DBD statement corresponds to the DBD statement:

```
*DDI DBD NAME=dbname
```

The REC statement corresponds to the SEGM statement/copylib:

```
*DDI REC NAME=cpylibrec | [new-COBOL-recordname],
*DDI      SEG=segname, [COPY=membername | [new-copylibname] ],
*DDI      [GEN01=Y | N]
```

The FLD statement corresponds to the FIELD and XDFLD statement/copylib field:

```
*DDI FLD NAME=cpylibfldname, IMSNAME=fldname,
*DDI      [PIC=cpylibpic]
```

The PSB statement corresponds to the PSB name:

```
*DDI PSB NAME=PSBname
```

The PCB statement is a positional, or placeholder statement. It corresponds to the PCB for which you are assigning an additional set of names:

```
*DDI PCB
```

VSAM DB

The VSM statement corresponds to the VSAM file external *ddname*:

```
*DDI VSM DDN=extddname, TYPE=K|E|R, VSPREFIX=fileprefix,
          [CYL|TRK|REC(size)], CISZ(cntlintrvalsize),
          VOL(volname), CAT=catname, [idcamsparms]
```

The REC statement corresponds to the copylib record name and copylib file name:

```
*DDI REC NAME=recname, SHORT=shrtrecname, COPY=membername,
          SOURCE=P|C|, MAXLEN=maxreclen, AVGLEN=avgreclen
```

The IDX statement is for a keyed file. It corresponds to copylib key field name and copylib file name:

```
*DDI IDX NAME=cpylibreckeyfldname, [ALIAS=redefinefld],
          TYPE=P|U|D, KEYLEN=keyfldlen, OFFSET=position,
          DDN=indxddname, PIC=number, [idcamsparms]
```

The SUB statement identifies the copylib record name(s) used by your APS program, record processing, batch options, and VSAM file external *ddname*:

```
*DDI SUB NAME=subschemaname, RECORD=recname,
        PROCOPT=A|G|I|D|R, [ACCESS=options], [BLOCK=0],
        [LABEL=STANDARD], [ASSIGN=extddname]
```

Parameters:	<i>ACCESS=options</i>	Batch access options. Dynamic for KSDS files. Sequential for ESDS/RRDS files.
	<i>ALIAS=redefinefld</i>	If a VSAM file uses multiple copylib records, this value is the field name that redefines <i>NAME=cpylibreckeyfldname</i> . Maximum 30 characters.
	<i>ASSIGN=extddname</i>	VSAM file external <i>ddname</i> . Default is the value of the *DDI VSM statement DDN parameter. Maximum 8 characters.
	<i>AVGLEN=avgreclen</i>	Average record length. Default is the value of MAXLEN.
	<i>CAT=catname</i>	Catalog name.
	<i>CISZ(cntlintrvalsize)</i>	Control interval size. Default is CISZ(4096).
	<i>COPY=membername</i>	Copylib member name. Default for IMS is SEGM NAME=. Maximum 8 characters.
	<i>CYL TRK REC(size)</i>	Size of file in cylinders, tracks or records.
	<i>DDN=extddname</i>	VSAM file external <i>ddname</i> . Maximum 8 characters.
	<i>DDN=indxddname</i>	DDNAME of the index. Default is <i>primary=ddname</i> . Maximum 24 characters.
	<i>GEN01=Y N</i>	GEN01=Y Indicates segname is an 01 level name. GEN01=N Overrides the value of &GEN-DB-REC-01-NAMES flag.

<i>idcamskeywrds</i>	Other IDCAMS keywords. Code exactly as they appear on IDCAMS control statements.
<i>IMSNAME=fieldname</i>	DBD value of FIELD NAME= or XDFLD NAME=. Maximum 8 characters.
<i>IMSNAME=new-COBOL fldname</i>	New copylib name for the new COBOL record .
<i>KEYLEN=keyfldlen</i>	Length of key field. Maximum 4 characters.
<i>MAXLEN=maxreclen</i>	Maximum record length.
<i>NAME=copylibfldname</i>	Copylib field name. Default is DBD value of FIELD NAME= or XDFLD NAME=. Maximum 30 characters.
<i>NAME=cpylibrec</i>	Copylib record name. Default is DBD value of SEGM NAME=. Maximum 30 characters.
<i>NAME=cpylibreckey fldname</i>	VSAM file copylib record key field name. Maximum 30 characters.
<i>NAME=dbname</i>	DBD name. Maximum 8 characters.
<i>NAME=new-COBOL-fldname</i>	New COBOL record name of the segment.
<i>NAME=new-COBOL-recname</i>	New COBOL record name of the segment.
<i>NAME=PSBname</i>	Specifies the program PSB name.
<i>NAME=recname</i>	VSAM file record name. Maximum 30 characters.
<i>NAME=subschemaname</i>	A unique subschema name. Maximum 8 characters.
<i>OFFSET=position</i>	The offset position of the field relative to the beginning of the record (first position is 0). Maximum characters.
<i>PIC=cpylibpic</i>	COBOL picture clause. Default is x(n). n=the value of the BYTES parameter in the DBD. Maximum 24 characters.

PIC= <i>keypicture</i>	Picture of keyed field. Required for non-alphanumeric. Maximum 24 characters.
PROCOPT=A G I D R	Process control options. Values are: A(I); G(et); I(nsert); D(elete); R(eplace).
RECORD= <i>recname</i>	Value of the first or only DDI REC statement NAME= <i>recname</i> . Maximum 30 characters.
SEG= <i>segname</i>	DBD value of SEGM NAME=. If you supplement or override this statement <i>segname</i> is as it appears in the program PSB. Maximum 8 characters.
SHORT= <i>shrtrecname</i>	Short record name. Maximum 8 characters.
SOURCE=P C	Method used to put copylib members in program. Values are: P (via an APS %INCLUDE statement). C(via COBOL COPY command).
TYPE=K E R	VSAM file type. Values are: K(eyed); E(ntry); R(elative). Default is K.
TYPE=P U D	Index type. Values are P(rietary); U(nique); D(uplicate). The first index must be primary.
VOL(<i>name</i>)	Volume name.
VSPREFIX= <i>fileprefix</i>	VSAM file prefix.

Comments:**IMS DB/DC**

- When APS generates DDI symbols, it converts special characters that appear in IMS data names follows:
 - \$ converts to an X
 - # converts to an Y
 - @ converts to a Z
- To access the index database itself, define the index DBD to the APS Generator, and include PCBs for it in PSBs defined to the APS Generator.

- If a PSB contains multiple PCBs for the database, use the VIEW or PCB parameter when you code the database macro to ensure that the correct PCB is used for the generated call.
- Select option 3, Generate DDISYMB symbols from DDIFILE, only if you want to regenerate DDI symbols that have been previously generated by following the procedures above.

Examples: IMS DB

DBD and copylib DDI statements.

```
DBD      NAME=DBD1 etc.
SEGM     NAME=S1 etc.
FIELD    NAME=S1FLD etc.
SEGM     NAME=S2 etc.
FIELD    NAME=S2FLD etc.
```

Copylib file (S1)

```
01 SALES-REC.
   05 REGION-FLD          PIC 9(03).
   .
   .
```

Copylib file (S2)

```
01 CUST-REC.
   05 LOCATION-FLD       PIC X(30).
   .
   .
```

DDI statements applicable to the above DBD statements and copylib.

```
KYWD 12-+----20-+----30-+----40-+----50-+---+
*DDI DBD NAME=DBD1
*DDI REC NAME=SALES-REC,SEG=S1,COPY=S1,
*DDI      GEN01=Y
*DDI FLD NAME=REGION-FLD,IMSNAME=S1FLD,PIC=9(03)
*DDI REC NAME=CUST-REC,SEG=S2,COPY=S2,
*DDI      GEN01=Y
*DDI FLD NAME=LOCATION-FLD,IMSNAME=S2FLD,PIC=X(30)
```

Fast path DBDs and PSBs.

```
DBD      NAME=DEDBASE,ACCESS=DEDB,RMNAME=DEDBRAND
AREA     DEVICE=3380,SIZE=1024,UOW=(20,5),ROOT=(40,10),DD1=AREA1
AREA     DEVICE=3380,SIZE=1024,UOW=(20,5),ROOT=(40,10),DD1=AREA2
AREA     DEVICE=3380,SIZE=1024,UOW=(20,5),ROOT=(40,10),DD1=AREA3
SEGM     NAME=CUST,PARENT=0,BYTES=(80,80)
FIELD    START=03,BYTES=06,TYPE=C,NAME=(ACCTNO,SEQ,U)
```

```

FIELD    START=09,BYTES=09,TYPE=C,NAME=SSA
FIELD    START=18,BYTES=30,TYPE=C,NAME=NAME
FIELD    START=48,BYTES=30,TYPE=C,NAME=ADDRESS
SEGM     NAME=SUMM,PARENT=CUST,BYTES=(30,30),TYPE=SEQ
FIELD    START=03,BYTES=10,TYPE=C,NAME=ACTION
FIELD    START=13,BYTES=08,TYPE=C,NAME=AMOUNT
FIELD    START=21,BYTES=10,TYPE=C,NAME=TELLER
SEGM     NAME=TRAN,PARENT=CUST,BYTES=(50,50),TYPE=DIR,      X
          SSPTR=8
FIELD    START=03,BYTES=06,TYPE=C,NAME=(TRANDATE,SEQ)
FIELD    START=09,BYTES=06,TYPE=C,NAME=TRANAMT
FIELD    START=15,BYTES=15,TYPE=C,NAME=TYPE
FIELD    START=30,BYTES=08,TYPE=C,NAME=BALANCE
SEGM     NAME=CHARGE,PARENT=TRAN,BYTES=(30,30),TYPE=DIR,SSPTR=5
FIELD    START=03,BYTES=06,TYPE=C,NAME=DATE
FIELD    START=09,BYTES=06,TYPE=C,NAME=SERVCHAR
FIELD    START=15,BYTES=10,TYPE=C,NAME=TYPECHAR
DBDGEN
FINISH
END

```

DDI statements:

```

*DDI DBD NAME=DEDBASE
*DDI REC NAME=CUSTOMER-RECORD,SEG=CUST,COPY=CUST
*DDI FLD NAME=CUSTOMER-ACCT-NO,IMSNAME=ACCTNO
*DDI FLD NAME=CUSTOMER-SSA,IMSNAME=SSA
*DDI FLD NAME=CUSTOMER-ADDRESS,IMSNAME=ADDRESS
*DDI REC NAME=SUMMARY-RECORD,SEG=SUMM,COPY=SUMM
*DDI FLD NAME=SUMMARY-ACTION,IMSNAME=ACTION
*DDI FLD NAME=SUMMARY-AMOUNT,IMSNAME=AMOUNT
*DDI FLD NAME=SUMMARY-TELLER,IMSNAME=TELLER
*DDI REC NAME=TRANSACTION-RECORD,SEG=TRAN,COPY=TRAN
*DDI FLD NAME=TRANSACTION-TRANDATE,IMSNAME=TRANDATE
*DDI FLD NAME=TRANSACTION-AMOUNT,IMSNAME=TRANAMT
*DDI FLD NAME=TRANSACTION-TYPE,IMSNAME=TYPE
*DDI FLD NAME=TRANSACTION-BALANCE,IMSNAME=BALANCE
*DDI REC NAME=CHARGE-RECORD,SEG=CHARGE,COPY=CHARGE
*DDI FLD NAME=CHARGE-DATE,IMSNAME=DATE
*DDI FLD NAME=CHARGE-SERVCHAR,IMSNAME=SERVCHAR
*DDI FLD NAME=CHARGE-TYPECHAR,IMSNAME=TYPECHAR

```

```

DBD      NAME=DEDB1,ACCESS=DEDB,
          RMNAME=(DBFHD040)
AREA     DD1=DEDB1DD,DEVICE=3380,SIZE=1024,
          ROOT=(10,5),UOW=(15,10)
SEGM     NAME=A,BYTES=(48,27),PARENT=0
FIELD    NAME=(A1,SEQ,U),BYTES=10,START=3,TYPE=C

```

Reference

```

SEGM    NAME=B,BYTES=( 24,11 ),PARENT=( ( A,SNGL ) ),TYPE=DIR,SSPTR=5
FIELD   NAME=( B1,SEQ,U ),BYTES=5,START=3,TYPE=C
FIELD   NAME=B2,BYTES=5,START=10,TYPE=C
SEGM    NAME=C,BYTES=( 34,32 ),PARENT=( ( B,DBLE ) ),RULES=( ,HERE ),
        TYPE=DIR
FIELD   NAME=( C1,SEQ,U ),BYTES=20,START=3,TYPE=C
SEGM    NAME=D,BYTES=( 52,33 ),PARENT=( ( A,DBLE ) ),TYPE=DIR,SSPTR=3
FIELD   NAME=( D1,SEQ,U ),BYTES=2,START=3,TYPE=C
SEGM    NAME=E,BYTES=( 52,33 ),PARENT=( ( A,DBLE ) ),RULES=( ,FIRST ),
        TYPE=DIR
FIELD   NAME=( E1,SEQ,U ),BYTES=2,START=3,TYPE=C
DBDGEN
FINISH
END

```

```
DDI *DDI DBD NAME=DEDB1
```

```

DBD     NAME=MSDB1,ACCESS=MSDB
DATASET REL=NO
SEGM    NAME=A,BYTES=4
FIELD   NAME=( A1,SEQ,U ),BYTES=1,START=1,TYPE=X
DBDGEN
FINISH
END

```

```
*DDI DBD NAME=MSDB1
```

```

PSB
PCB     TYPE=DB,DBDNAME=DEDB1,PROCOPT=A,KEYLEN=35
SENSEG  NAME=A,PARENT=0
SENSEG  NAME=B,PARENT=A,SSPTR=( ( 1,R ), ( 2,U ), ( 5 ) )
SENSEG  NAME=C,PARENT=B
PCB     TYPE=DB,DBDNAME=DEDBASE,PROCOPT=A,KEYLEN=12
SENSEG  NAME=CUST,PARENT=0
SENSEG  NAME=SUMM,PARENT=CUST
SENSEG  NAME=TRAN,PARENT=CUST
PCB     TYPE=DB,DBDNAME=MSDB1,PROCOPT=A,KEYLEN=1
SENSEG  NAME=A,PARENT=0
PSBGEN  PSBNAME=FASTPATH,LANG=COBOL
END

```

- Copylib for a fixed length KSDS file with three indices accessing one record.

```

01      EMPLOYEE-RECORD.
        05      EMPL-EMP-NUM                PIC X(06) .
        05      EMPL-LAST-NAME             PIC X(20) .
        05      EMPL-FIRST-INIT           PIC X(02) .

```

```

05     EMPL-MIDDLE-INIT           PIC X(02) .
05     EMPL-SOURCE-CODE          PIC X(06) .
05     EMPL-SALARY               PIC 9(7)COMP-3 .
05     EMPL-SCHED-HOURS         PIC 9(04) .
05     EMPL-ACTIVE-FLAG         PIC X(02) .
05     EMPL-LAST-PROM-DATE      PIC X(06) .
05     EMPL-LAST-EXT-DATE      PIC X(06) .
05     EMPL-START-YR           PIC X(02) .

```

DDI for fixed length KSDS file.

```

KYWD 12-+-----20---+-----30---+-----40---+-----50---+-----
*DDI VSM DDN=EMPLOYEE
*DDI     TYPE=K, VSPREFIX=VQAC6550.APS17X,
*DDI     TRK(100 10), CISZ(8192), VOL(SAGE03),
*DDI     REPLICATE, FSPC(15 15), SHR(1 3), NOIMBED
*DDI REC NAME=EMPLOYEE-RECORD, SHORT=EMPL,
*DDI     COPY=EMPLOYEE, SOURCE=P,
*DDI     MAXLEN=56
*DDI IDX NAME=EMPL-EMP-NUM, TYPE=P, KEYLEN=6, OFFSET=0,
*DDI     DDN=EMPLOYEE, TRK(5 5), VOL(SAGE03)
*DDI IDX NAME=EMPL-SOURCE-CODE, TYPE=D,
*DDI     KEYLEN=6, OFFSET=30, DDN=EMPLOYEE1,
*DDI     TRK(3 3), VOL(SAGE03), NOIMBED
*DDI IDX NAME=EMPL-SALARY, TYPE=D,
*DDI     KEYLEN=4, OFFSET=36, DDN=EMPLOYEEZ,
*DDI     PIC=9(7)COMP-3, TRK(3 3), VOL(SAGE03)
*DDI SUB NAME=SAMPPGM, RECORD=EMPLOYEE-RECORD, PROCOPT=A,
*DDI     LABEL=STANDARD, BLOCK=200, ASSIGN=EMPLOYEE

```

Copylib for a variable length keyed sequential file with one index accessing multiple records.

```

% * CICS RECORDS ARE REDEFINED FOR WORKING-STORAGE.
% * MVS RECORDS ARE NOT REDEFINED; ARE PLACED DIRECTLY UNDER
FD.
% IF &APS-MDC = "CICS-TP"
  &VSSUF = " REDEFINES ORDER-RECORD"
% ELSE
  &VSSUF = ""
01 ORDER-RECORD.
05 ORDER-KEY           PIC X(05) .
05 ORDER-RECORD-TYPE  PIC X(01) .
05 ORDER-CUST-NUMBER  PIC X(07) .
05 ORDER-PART-NUMBER  PIC X(07) .
05 ORDER-QUANTITY-ORDERED  PIC 9(05) .
05 ORDER-QUANTITY-TYPE  PIC X(10) .
05 ORDER-ORDER-AMOUNT  PIC 9(05)V99 .

```

```

    05 ORDER-ORDER-STATUS          PIC X(04).
01 ORDER-PART-RECORD &VSSUF.
    05 FILLER                      PIC X(06).
    05 PART-NAME                   PIC X(05).
    05 PART-DESCRIPTION            PIC X(25).
    05 PART-SUPPLIER-NBR          PIC X(07).
    05 PART-SUPPLIER-NAME         PIC X(25).
01 ORDER-DELV-RECORD &VSSUF.
    05 FILLER                      PIC X(06).
    05 DELV-CONTACT-NAME          PIC X(30).
    05 DELV-CONTACT-PHONE         PIC X(12).
    05 DELV-ADDRESS               PIC X(35).
    05 DELV-SPECIAL-INSTRUCTIONS  PIC X(50).

```

DDI for variable length KSDS files.

```

KYWD 12-+-----20-+-----30-+-----40-+-----50-+-----
*DDI VSM DDN=ORDER
*DDI REC NAME=ORDER-RECORD, SHORT=ORDER, COPY=ORDER, SOURCE=P,
*DDI     MAXLEN=133, AVGLEN=46
*DDI IDX NAME=ORDER-KEY, TYPE=P, KEYLEN=5, OFFSET=0, DDN=ORDER
*DDI REC NAME=ORDER-PART-RECORD, SHORT=PART, MAXLEN=68
*DDI REC NAME=ORDER-DELV-RECORD, SHORT=DELV, MAXLEN=133
*DDI SUB NAME=ORDERSS,
*DDI     RECORD=ORDER-RECORD, PROCOPT=A,
*DDI     LABEL=STANDARD, BLOCK=0, ASSIGN=ORDER

```

DDI statements for generating DDISYMB and IDCAMS.

```

*DDI VSM DDN=PERSON
*DDI     TYPE=K, VSPREFIX=VAPS6550,
*DDI     CISZ(4096), VOL(PDVL02), TRK(5 5)
*DDI REC NAME=PERSONNEL-RECORD, SHORT=PERSONNEL,
*DDI     COPY=PERSONNEL, SOURCE=P, MAXLEN=80, PREFIX=PER
*DDI IDX NAME=SSA, ALIAS=SSA-X, TYPE=P, KEYLEN=9,
*DDI     OFFSET=0, DDN=PERSON,
*DDI     TRK(5,5), VOL(PDVL02)
*DDI IDX NAME=LAST-NAME, TYPE=D, KEYLEN=15,
*DDI     OFFSET=9, DDN=PERSON1,
*DDI     TRK(5,5), REUSE, VOL(PDVL02)
*DDI IDX NAME=TITLE, TYPE=D, KEYLEN=15,
*DDI     OFFSET=24, DDN=PERSON2,
*DDI     TRK(5,5), REUSE, VOL(PDVL02)
*DDI SUB NAME=SAMPLE, RECORD=PERSONNEL-RECORD,
*DDI     PROCOPT=A

```

Generated IDCAMS source.

03590000

DELETE VAPS6550.PERSON.CLUSTER	01550000
CLUSTER	01610000
PURGE	01620000
	01640000
IF LASTCC = 8	01650000
THEN	01660000
SET LASTCC = 00	01670000
	01680000
DEFINE CLUSTER	01690000
(INDEXED	01760000
NAME (VAPS6550.PERSON.CLUSTER)	01820000
UNIQUE)	01980000
	01990000
DATA	02000000
(NAME (VAPS6550.PERSON.DATA)	02010000
TRK(5,5)	02030000
VOL(PDVL02)	02080000
RECSZ (80 80)	02130000
CISZ (4096)	02170000
KEYS (9 0)	02240000
SHR (3 3))	02320000
	02340000
INDEX	02350000
(NAME (VAPS6550.PERSON.INDEX)	02360000
TRK(5,5)	02440000
VOL(PDVL02)	02460000
CISZ (1024)	02550000
SHR (3 3)	02660000
)	02690000
	02710000
IF LASTCC < 5	02720000
THEN	02730000
REPRO	02740000
INFILE (DD1)	02750000
ODS (VAPS6550.PERSON.CLUSTER)	02760000
	02770000
DEF AIX	02890000
(NAME (VAPS6550.PERSON1.AIX)	02990000
RELATE (VAPS6550.PERSON.CLUSTER)	03000000
REUSE	03040000
TRK(5,5)	03110000
VOL(PDVL02)	03180000
RECSZ (256 512)	03220000
CISZ (1024)	03260000
KEYS (15 9)	03290000
NONUNIQUEKEY	03340000
UPGRADE)	03380000
DATA	03390000

(NAME (VAPS6550.PERSON1.DATA))	03400000
INDEX	03410000
(NAME (VAPS6550.PERSON1.INDEX))	03420000
	03430000
IF LASTCC < 5	03440000
THEN	03450000
BLDINDEX	03460000
IDS (VAPS6550.PERSON.CLUSTER)	03470000
ODS (VAPS6550.PERSON1.AIX)	03480000
	03490000
IF LASTCC < 5	03500000
THEN	03510000
DEF PATH	03520000
(NAME (VAPS6550.PERSON1.PATH)	03530000
PENT (VAPS6550.PERSON1.AIX)	03540000
UPDATE)	03550000
DEF AIX	02890000
(NAME (VAPS6550.PERSON2.AIX)	02990000
RELATE (VAPS6550.PERSON.CLUSTER)	03000000
REUSE	03040000
TRK(5,5)	03110000
VOL(PDVL02)	03180000
RECSZ (256 512)	03220000
CISZ (1024)	03260000
KEYS (15 24)	03290000
NONUNIQUEKEY	03340000
UPGRADE)	03380000
DATA	03390000
(NAME (VAPS6550.PERSON2.DATA))	03400000
INDEX	03410000
(NAME (VAPS6550.PERSON2.INDEX))	03420000
	03430000
IF LASTCC < 5	03440000
THEN	03450000
BLDINDEX	03460000
IDS (VAPS6550.PERSON.CLUSTER)	03470000
ODS (VAPS6550.PERSON2.AIX)	03480000
	03490000
IF LASTCC < 5	03500000
THEN	03510000
DEF PATH	03520000
(NAME (VAPS6550.PERSON2.PATH)	03530000
PENT (VAPS6550.PERSON2.AIX)	03540000
UPDATE)	03550000

DDISYMB Flags

Compatibility: SQL target

Description: You can use special DDISYMB flags to suppress or modify DCLGEN copybook generation. After you generate your subschema and DDISYMB file, insert these override flags into the DDISYMB file.

Syntax: Flag 1, suppress inclusion of the APS-generated DCLGEN copybook:

```
&D2-INCLUDED-COPYLIB-copybookname = "YES" | "NO"
```

Flag 2, specify user override of APS indicator structure generation:

```
&D2-INCLUDED-IV-copybookname = "YES" | "NO"
```

Flag 3, specify indicator variable prefix override, overriding the APS default prefix IND:

```
&D2-copybookname-IV-PREFIX = "prefixvalue"
```

Flag 4, specify indicator structure, group-level, override name:

```
&D2-copybookname-IV-01-NAME = "indicatorvariablename"
```

Flag 5, specify host structure, group-level override name:

```
&D2-copybookname-HOST-01-NAME = "01levelname"
```

Flag 6, generate a value for &D2-*copybookname*-HOST-01-NAME:

```
&D2-GLOBAL-DCLGEN-NAME = "YES" | "NO"
```

Flag 7, override use and generation of the default IND-cursorname structure:

```
&D2-USE-CURSOR-IND = "YES" | "NO"
```

- Comments:**
- Code Flag 1 to:
 - Include your own DB2 host variable copybook.
 - Suppress inclusion of the corresponding APS-generated DCLGEN copybook named by *copybookname*.

- Code Flag 2 to:
 - Include your own indicator variable structure in the copybook or DB2 DDI symbols for that table.
 - Suppress generation of the APS host indicator variable structure for the table named by copybookname.
- For Flag 2, the APS-generated host indicator variable structure follows the format:

```
01  IND-copybookname-REC
    05  IND-colname1
    05  IND-colname2
```

If you override the host indicator variable structure generated by APS, you still must conform to the format above. However, you can override the IND prefix with one of your own. You can specify one prefix for each host indicator variable structure. The prefix you create must be supplied to APS via the variable &D2-copybookname-IV-prefix.

Similarly, you may also override the 01-level name of the host indicator variable structure. If the 01-level name you choose is not in the APS format shown above, supply it to APS via the variable &D2-copybookname-IV-01-name. These two user flags are described below.

- For Flag 3:
 - Code it to substitute a value for the default prefix, IND, on all indicator variables for a table.
 - Include this flag to supply your own indicator variable structure and if the variable names have a prefix other than IND.
 - The prefix you enter can include up to 11 characters.
 - Follow the naming convention of prefix-columnname for indicator variable names, where prefix is either IND or a value you specify in the prefix override. For example, for a table represented by a copybook named TAB2, the entry:

```
% &D2-TAB2-IV-PREFIX = "TAB2-IV"
```

Yields the following indicator variable structure.

```
01  TAB2-IV-TAB2-REC.
    05  TAB2-IV-colname1
```

```

05  TAB2-IV-colname2
05  TAB2-IV-colname3
.
.
05  TAB2-IV-colnameN

```

- If you do not supply this variable, the APS uses the structure generated for each cursor, *cursorname-IND-variable*, for cursor processing.
- For Flag 4:
 - Code it to create your own 01-level name for the indicator variable structure. *Indicatorvariablename* replaces the APS-generated 01-level name. The default for the 01-level name is *prefixcolumnamerec*.
 - Include this flag to supply an override indicator variable structure when the 01-level name does not conform to the APS naming convention.
 - The maximum length of *indicatorvariablename* is 30 characters.
 - Indicator variable names must follow the naming convention of *prefix-columnname*, in which *prefix* is either IND or a value you specify in *&D2-copybookname-IV-PREFIX*. For example, a table represented by a copybook named TAB2, the entry:

```
% &D2-TAB2-IV-01-NAME = "TAB2-IND-VAR"
```

Yields the following indicator variable structure:

```

01  TAB2-IND-VAR.
    05  IND-colname1
    05  IND-colname2
    05  IND-colname3
    .
    .
    05  IND-colnameN

```

- If you do not supply this variable, the APS uses the structure generated for each cursor, *cursorname-IND-variable*, for cursor processing.
- Code Flag 5 to change the default host structure after the DCLGEN process creates it. *01levelname* replaces the APS-generated 01-level name, *copylibnamerec*. APS uses your host 01-level name when it qualifies host variables during SQL generation.

- You can use Flag 6, rather than Flag 5, to generate a name for all your tables. In the APS CNTL file APDB2IN, set the flag &D2-GLOBAL-DCLGEN-NAME to yes. APS generates a value for &D2-copybookname-HOST-01-NAME, using the IBM DCLGEN convention of DCLtablename. Defining this flag in APDB2IN overrides manually-coded definitions in all applications.
- Use Flag 7 to override the default indicator variable structure, IND-cursorname with the structure IND-tablerec. APS generates the default indicator variable structure, as follows:

```

01  TABLE-REC .
    05  COL1                PIC X(4) .
    05  COL2                PIC X(8) .
01  IND-TABLE-REC .
    05  IND-COL1           PIC S9(4) COMP .
01  IND-CURSOR1 .
    05  IND-COL1           PIC S9(4) COMP .
    .
    .
    .
    DB-FETCH CURSOR CURSOR1
    IF OK-ON-REC
        IF IND-COL1 OF IND-CURSOR1 = +0
            MAP-COL1 = COL1 in TABLE-REC

```

Note: Using the host indicator variable structure can simplify your code because the one indicator variable structure can be used repeatedly. In nested processes, however, the host indicator variables are overwritten by each successive nesting level.

Examples: Substitute the prefix TAB2-IV for the default prefix IND for all indicator variables in a table represented by copybookname and for the 01-level name.

```
% &D2-copybookname-IV-PREFIX = "TAB2-IV"
```

Substitute a different 01-level name in the indicator variable structure generated by APS.

```
% &D2-copybookname-IV-01-NAME = "USER-IND-VAR-NAME"
```

Sample source code:

```
% &D2-D2MASTER-HOST-01-NAME = "XYZ-D2MASTER-STRUCT"
```

```

*****
*   COBOL DECLARATION FOR TABLE D2MASTER
*   01-LEVEL NAME POST-PROCESSED
*****
01  XYZ-D2MASTER-STRUCT.
    10  PM-PART-NO                PIC X(8).
    10  PM-NEW-PART-NO            PIC X(8).
    10  PM-OLD-PART-NO            PIC X(8).
    10  PM-PART-SHORT-DESC        PIC X(13).
    10  PM-UNITS                  PIC 9(5).
    10  PM-UNIT-BASE-PRICE        PIC 9(5).
    10  PM-DIMENSIONS             PIC X(8).
    10  PM-COLOR                  PIC X(8).

```

Generated SQL code using *&D2-D2MASTER-HOST-01-NAME*

```

EXEC SQL SELECT ...
INTO XYZ-D2MASTER-STRUCT.PM-PART-NO,
XYZ-D2MASTER-STRUCT.PM-NEW-PART-NO,
      .
      .
      .
      XYZ-D2MASTER-STRUCT.PM-COLOR,
FROM ...
WHERE ...
END-EXEC

```

Generated SQL code without using *&D2-D2MASTER-HOST-01-NAME*

```

EXEC SQL SELECT ...
INTO D2TAB-REC.PM-PART-NO,
      D2TAB-REC.PM-NEW-PART-NO,
      .
      .
      .
      D2TAB-REC.PM-COLOR,
FROM ...
WHERE ...
END-EXEC

```

DECL

Category: Program Painter and Specification Editor parameter

Compatibility: ISPF Dialog and ISPF prototyping batch programs; CICS and IMS DC programs without screens

Description: Create Declarative Section statements only--not sections or paragraphs.

See also *DPAR* and *USE BEFORE REPORTING* for creating Declarative Section statements, paragraphs, and sections.

Syntax: -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
DECL *declarativestements*

- Comments:**
- We recommend that you code declaratives at the end of your program, because APS generates the END DECLARATIVES statement when either:
 - It encounters another keyword in the KYWD *column*.
 - The Declaratives Section is at the end of the program.
 - Do not code the DECLARATIVE SECTION header or the END DECLARATIVES statement. APS generates these for you.

Example: -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
DECL *declarative statement*
declarative statement

Generated APS source:

```
DECLARATIVES .
declarative statement
declarative statement
END DECLARATIVES
```

DLG-ISPEXEC

Category: Data communication call (see *Data Communication Calls*)

Compatibility: ISPF Dialog target

Description: Invoke ISPF services that use CALL ISPEXEC format.

Syntax: DLG-ISPEXEC *commandproceduresyntax*

Parameters: *commandproceduresyntax* Syntax for call executed in Command Procedure (CLIST) format.

Comment: Data field DLG-ISPEXEC-RC contains the return code after the call execution.

Examples: Use the CONTROL service to disable the user's split screen capability.

```
DLG-ISPEXEC CONTROL SPLIT DISABLE
IF DLG-ISPEXEC-RC = 8
  /* SPLIT SCREEN ALREADY DISABLED
  TRUE SPLIT-SCREEN-DISABLED
ELSE-IF DLG-ISPEXEC-RC = 0
  /* SPLIT SCREEN DISABLED
  TRUE SPLIT-SCREEN-DISABLED
```

Define an application message library to search before the default message library.

```
DLG-ISPEXEC LIBDEF ISPMLIB DATASET ID('ABC.DEF.ISPMLIB')
IF DLG-ISPEXEC-RC = 0
  /* OK
ELSE
  DISPLAY '???' LIBDEF ISPMLIB ERROR, RC = ' DLG-ISPEXEC-RC
```

Invoke a command procedure to allocate application files.

```
DLG-ISPEXEC SELECT CMD(%ALOCFILE ALLOC)
IF DLG-ISPEXEC-RC = 0
  /* OK
ELSE
  DISPLAY '???' ALOCFILE ERROR, RC = ' DLG-ISPEXEC-RC
```

DLG-ISREDIT

Category: Data communication call (see *Data Communication Calls*)

Compatibility: ISPF Dialog target

Description: Invoke ISREDIT services.

Syntax: `DLG-ISREDIT commandproceduresyntax`

Parameter: *commandproceduresyntax* Syntax for call executed in Command Procedure (CLIST) format

Comment: Data field DLG-ISREDIT-RC contains the return code after call execution.

Example: Determine if the member is new or existing. DLG-VDEFINE links ISPF variable &LASTLINE to COBOL variable LASTLINE.

```
DLG-VDEFINE LASTLINE &APS-FULL
DLG-ISREDIT (LASTLINE) = LINENUM .ZLAST
IF LASTLINE = 0
  /* NEW MEMBER
  TRUE NEW-MEMBER
```

DLG-SETMSG

Category: Data communication call (see *Data Communication Calls*)

Compatibility: ISPF Dialog target

Description: Display a message on the next panel; predefine message text and attributes.

Syntax: Format 1, SETMSG definition:

```
DLG-SETMSG
... [SHORT 'shortmessagetext']
... [LONG 'longmessagetext']
```

```
... [ALARM 'YES' | 'NO' ]
... [HELP 'helppanelname' ]
```

Format 2, SETMSG definition for execution by Format 3:

```
DLG-SETMSG erroridentifier
... [SHORT 'shortmessagetext' ]
... [LONG 'longmessagetext' ]
... [ALARM 'YES' | 'NO' ]
... [HELP 'helppanelname' ]
```

Format 3, SETMSG execution

```
DLG-SETMSG [erroridentifier|messageID]
```

Parameters:	ALARM 'YES' 'NO'	Sound alarm when screen displays.
	<i>erroridentifier</i>	Unique name referencing predefined message text and attributes.
	HELP ' <i>help panelname</i> '	Help panel that displays via PF1.
	<i>messageID</i>	ISPF MSGID for SETMSG service.
	LONG ' <i>long messagetext</i> '	Display text in long message field when the end user presses PF1 for the first time. Messagetext must fit on the same line as LONG.
	SHORT ' <i>short messagetext</i> '	Display text in short message field. Messages longer than 24 characters can cause truncation errors. Messagetext must fit on the same line as SHORT.

- Comments:**
- If you use Format 2, include at least one of the keywords.
 - Data field DLG-SETMSG-RC checks the return code after call execution.

Examples: Define a message for an invalid option condition. INVALID-OPT (error identifier) identifies the message information to display when DLG-SETMSG is invoked in the execution format.

```
DLG-SETMSG INVALID-OPT
... SHORT 'INVALID OPTION'
... LONG 'ENTER ONE OF THE LISTED OPTIONS'
... ALARM 'YES'
```

Then, display the invalid option message.

```
IF SCRA-OPTION = 'value'
  /* VALID CONDITION
ELSE
  DLG-SETMSG INVALID-OPT
```

DLG-VCOPY

Category: Data communication call (see *Data Communication Calls*)

Compatibility: ISPF Dialog target

Description: Copy data from a Dialog variable to a COBOL program variable and generate the Working-Storage entry for the COBOL variable, if the COBOL level is specified.

Syntax: DLG-VCOPY [*COBOLlevel*] *COBOLvariable*
 ... [[FROM] *dialogvariable*]
 ... [PIC *COBOLpicture*] | LEN *value*
 ... [GENONLY]

Parameters:	COBOLlevel	COBOLvariable level number.
	COBOLvariable	COBOL data name the call processes.
	FROM <i>dialog variable</i>	ISPF Dialog variable where data comes from; default is COBOLvariable, truncated to eight characters.
	GENONLY	Define COBOLvariable data item in Working-Storage only.
	LEN <i>value</i>	COBOLvariable length; can be numeric integer, COBOL variable, or arithmetic expression.
	PIC <i>COBOLpicture</i>	COBOLvariable picture; default is alphanumeric.

Comments:

- Code FROM dialogvariable on the same line as COBOLvariable.
- Coding COBOLlevel generates COBOLvariable Working-Storage; otherwise, define the COBOL variable in Working-Storage.

- Data field DLG-VCOPY-RC contains the return code after call execution.

Examples: Copy data from a system variable into a screen field. The call does not generate the COBOL variable because the COBOL level number is not coded.

```
DLG-VCOPY SCRA-ZUSER FROM ZUSER LEN 8
```

Copy data from a system variable into a Working-Storage variable. Coding the COBOL level number generates the COBOL variable.

```
DLG-VCOPY 01 WS-LONG-ERROR-MESSAGE PIC X(78) FROM ZERRLM
```

DLG-VDEFINE

Category: Data communication call (see *Data Communication Calls*)

Compatibility: ISPF Dialog target

Description: Establish a link between a Dialog function pool variable and a COBOL program variable.

Syntax: DLG-VDEFINE [*COBOLlevel*] *COBOLvariable*
 ... [[AS] *dialogvariable*]
 ... [PIC *COBOLpicture*] | LEN *value*
 ... [GENONLY]

Parameters:	<i>AS dialog variable</i>	ISPF Dialog variable where data links from; default is <i>COBOLvariable</i> , truncated to eight characters.
	<i>COBOLlevel</i>	<i>COBOLvariable</i> level number.
	<i>COBOLvariable</i>	COBOL data name the call processes.
	GENONLY	Define the <i>COBOLvariable</i> data item in Working-Storage only.

LEN *value* COBOLvariable length; can be numeric integer, COBOL variable, or arithmetic expression.

PIC *COBOLpicture* COBOLvariable picture; default is alphanumeric.

- Comments:**
- Code *AS dialogvariable* on the same line as COBOLvariable.
 - NTRY automatically links any panel variables to screen field names.
 - Coding *COBOLlevel* generates COBOLvariable in Working-Storage; otherwise, define the COBOL variable in Working-Storage.
 - Data field DLG-VDEFINE-RC checks the return code after call execution.

Example: Establish a link between function pool variable COMDATA and COBOL variable WS-COMM-DATA. Coding the COBOL level number generates the COBOL data name.

```
DLG-VDEFINE 01 WS-COMM-DATA PIC X(150) AS COMDATA
```

DLG-VDELETE

Category: Data communication call (see *Data Communication Calls*)

Compatibility: ISPF Dialog target

Description: Remove the Dialog variables, previously defined by VDEFINE, from the function pool.

Syntax: DLG-VDELETE *dialogvariable* | *

Parameters: *dialogvariable* Delete a specific ISPF Dialog variable.
* (asterisk) Delete all variables.

- Comments:**
- Coding an asterisk deletes the variables according to the value of the control variable &DLG-AUTO-VARIABLE-VDELETE.
 - Data field DLG-VDELETE-RC contains the return code after call execution.

Example: Remove the link between function pool variable COMDATA and COBOL variable WS-COMM-DATA.

```
DLG-VDEFINE 01 WS-COMM-DATA LEN(150)
... AS COMDATA
DLG-VDELETE COMDATA
```

DLG-VREPLACE

Category: Data communication call (see *Data Communication Calls*)

Compatibility: ISPF Dialog target

Description: Move data from a COBOL program variable to an ISPF function pool variable.

Syntax:

```
DLG-VREPLACE [COBOLlevel] COBOLvariable
... [[INTO] dialogvariable]
... [PIC COBOLpicture]|LEN value
... [GENONLY]
```

Parameters:	<i>COBOLlevel</i>	COBOLvariable level number.
	<i>COBOLvariable</i>	COBOL data name the call processes.
	GENONLY	Define COBOLvariable data item in Working-Storage only.
	INTO <i>dialog variable</i>	ISPF function pool variable data replaces; default is COBOLvariable, truncated to eight characters.
	LEN <i>value</i>	COBOLvariable length; can be numeric integer, COBOL variable, or arithmetic expression.
	PIC <i>COBOLpicture</i>	COBOLvariable picture; default is alphanumeric.

Comments:

- Code INTO dialogvariable on the same line as COBOLvariable.
- Coding COBOLlevel generates COBOLvariable in Working-Storage; otherwise, define the COBOL variable in Working-Storage.

Example: Move a new value to the function pool variable ZPF01 and invoke ISPF help services with PF01.

```
DLG-VREPLACE 01 WS-PF01-HELP PIC X(04) VALUE 'HELP' INTO ZPF01
```

DLG-VRESET

Category: Data communication call (see *Data Communication Calls*)

Compatibility: ISPF Dialog target

Description: Reset all program function pool variables and delete the links between COBOL variables and Dialog variables within the function pool.

Syntax: DLG-VRESET

Comment: Data field DLG-VRESET-RC contains the return code after call execution.

DPAR

Category: Program Painter and Specification Editor parameter (see *Keywords*)

Compatibility: ISPF Dialog and ISPF prototyping programs; CICS and IMS DC batch programs and reports

Description: Create a Declarative Section or section paragraph--not declarative statements.

See also *DECL* and *USE BEFORE REPORTING* for creating Declarative Section statements, paragraphs, and sections.

Syntax:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
DPAR   sectionname SECTION
        USE declarativesentence
[ DPAR   paragraphname
```

paragraphstatements]

Parameters: *sectionname* Specify Section paragraph.
 USE *declarative sentence* APS supports the USE clause with the exception of USE AFTER DEBUGGING, which is not supported.

Comments:

- We recommend that you code declaratives at the end of your program, because APS generates the END DECLARATIVES statement when either:
 - It encounters another parameter in the KYWD *column*.
 - The Declaratives Section is at the end of the program.
- Do not code the DECLARATIVE SECTION header and the END DECLARATIVES statements. APS generates these for you.

Example: Program Painter code:

```
-KYWD- 12-*---20---*---30---*---40---*---50---*---60---
*---70-
DPAR  DUMMY-FOOTER SECTION
      USE BEFORE REPORTING FOOTER-DUMMY
DPAR  DUMMY-FOOTER-PARA
      MOVE TOTAL-DIFF TO TIME-TOTAL
      SUPPRESS PRINTING
DPAR  TOTAL-FOOT-SECTION SECTION
      USE BEFORE REPORTING TOTAL-FOOT
      TOTAL-FOOT-PARA
      TIME-AVERAGE = TIME-TOTAL / AVERAGE-CNT
      CALL-PERCENTAGE = (HALF-HOUR-CALLS / AVERAGE-CNT)
      ... * 100
      MOVE HALF-HOUR-CALLS TO HOLD-CALLS
      ADD HOLD-CALLS TO HALF-HOUR-CNT
      MOVE ZERO TO HALF-HOUR-CALLS
DPAR  CONTROL-FOOTING-FINAL SECTION
      USE BEFORE REPORTING CNTL-FT-GP
DPAR  CONTROL-FOOTING-FINAL-PARA
      IF SYSIN-TRACKER NOT = 'CTSALL'
        SUPPRESS PRINTING
      ELSE
        FINAL-PERCENTAGE =
          (HALF-HOUR-CNT / FINAL-PROB-CNT * 100)
```

Generated code:

```

DECLARATIVES .

DUMMY-FOOTER SECTION.
    USE BEFORE REPORTING FOOTER-DUMMY
DUMMY-FOOTER-PARA .
    MOVE TOTAL-DIFF TO TIME-TOTAL
    SUPPRESS PRINTING
TOTAL-FOOT-SECTION SECTION.
    USE BEFORE REPORTING TOTAL-FOOT
    $TOTAL-FOOT-PARA
    TIME-AVERAGE = TIME-TOTAL / AVERAGE-CNT
    CALL-PERCENTAGE = (HALF-HOUR-CALLS / AVERAGE-CNT)
    ... * 100
    MOVE HALF-HOUR-CALLS TO HOLD-CALLS
    ADD HOLD-CALLS TO HALF-HOUR-CNT
    MOVE ZERO TO HALF-HOUR-CALLS
CONTROL-FOOTING-FINAL SECTION.
    USE BEFORE REPORTING CNTL-FT-GP
CONTROL-FOOTING-FINAL-PARA .
    IF SYSIN-TRACKER NOT = 'CTSALL'
        SUPPRESS PRINTING
    ELSE
        FINAL-PERCENTAGE =
            (HALF-HOUR-CNT / FINAL-PROB-CNT * 100)

END DECLARATIVES

```

DS

Category: Program Painter and Specification Editor parameter (see *Keywords*)

Description: In your program, include a data structure created in the Data Structure Painter and in that format.

Syntax: -KYWD- 12-*----20---*----30---*----40---*----50---*----60
DS[nn] *datastructurename*

Comments:

- *Nn* is the beginning level number for the data structure entity. This is useful when concatenating multiple data structure entities in the

same program. *Nn* overrides the default 01-level created with the REC parameter in the Data Structure Painter.

- The preceding section parameter determines the placement of a data structure in the generated program. Associated section keywords are

FD	File Section (see <i>FD</i>)
SD	Sort File Description (see <i>SD</i>)
WS	Working-Storage Section (see <i>WS</i>)
LK	Linkage Section (see <i>LK</i>)

Entity Content Report (MS02)

Category: APS-generated report (see *Application Reports*)

Description: The Entity Content Report lists the following information for each component of an application.

- The painter where you create the component
- The component name
- The date when the component was created
- The date when the component was last updated
- The title of the component

You can produce a report for one type of component, or all types. If you include all types, the report provides a separate section for components by painter, with the painters arranged alphabetically.

This report helps you track the status of an evolving application. Use it to verify which components have been created and modified as planned.


```

PROG  APP2          12/17/90   12/18/90
PROG  EVOM          01/10/91   01/10/91
PROG  EVPL          01/10/91   01/10/91
.
.
REPT  COSTRPT      07/27/90   07/27/90   *** NOT AVAILABLE ***
REPT  MERPT        08/24/90   08/24/90   *** NOT AVAILABLE ***
REPT  T1RPT        07/27/90   07/27/90   *** NOT AVAILABLE ***
SCRN  AAAAA        06/04/90   06/07/90   TEST SCREEN
SCRN  DLMENU       03/29/90   03/29/90
.
.
APPLICATIONS:    45
SCENARIOS:       43
DATA-STRUCTURES: 27
PROGRAMS:        93
REPORTS:         3
SCREENS:         112

```

Entity Cross Reference (MD01)

Category: APS-generated report (see *Application Reports*)

Description: The Entity Cross Reference Report provides a list of application components and the painters where you create the components. Use this report for impact analysis, when you need to find the components affected by a proposed change. For example, when a data structure changes, that can affect components in a variety of applications and programs. This report can show at a glance all of the affected components that reference a particular data structure.

The report has a section for each cross-referenced component. The report arranges the associated components in alphabetical order, along with the type of each component and a description of it. The report ends with the total number of cross-referenced components.

Comment: Produce the Entity Cross Reference report from the Documentation Facility.

Example:

```

REPORT CODE: MD01                APS APPLICATION DICTIONARY                PAGE      1
                                ENTITY CROSS REFERENCE                    01/17/92 08:48

MKTAPS.MKT2
SELECTION CRITERIA: PROGRAM
  ENTITY NAME = ADEMO
*****
ENTITY:  ADEMO                    CREATED:  09/17/90
TITLE:                                UPDATED:  09/18/90
*****

```

```

ASSOCIATED ENTITY                TYPE                TITLE
-----
*** NO ASSOCIATED ENTITIES FOUND FOR THIS SELECTION ***

```

```

REPORT CODE: MD01                APS APPLICATION DICTIONARY                PAGE     99
                                ENTITY CROSS REFERENCE                    01/17/92 08:48

MKTAPS.MKT2
SELECTION CRITERIA: PROGRAM
  ENTITY NAME = ALL
*****
ENTITY:  TDCM                    CREATED:  03/19/90
TITLE:                                UPDATED:  09/17/90
*****

```

```

ASSOCIATED ENTITY                TYPE                TITLE
-----
MVS21                            APSAPPL
TDDEMO                           APSAPPL
TDCM                             APSEXPS

```

```

REPORT CODE: MD01                APS APPLICATION DICTIONARY                PAGE    100
                                ENTITY CROSS REFERENCE                    01/17/92 08:48

MKTAPS.MKT2
SELECTION CRITERIA: PROGRAM
  ENTITY NAME = ALL
*****
ENTITY:  TDCS                    CREATED:  04/26/90
TITLE:                                UPDATED:  08/24/90
*****

```

```

ASSOCIATED ENTITY                TYPE                TITLE
-----
MVS21                            APSAPPL
TDDEMO                           APSAPPL
TDCS                             APSEXPS

```

```

TOTAL NUMBER OF DEFINED PROGRAMS -    60
TOTAL NUMBER OF UNDEFINED PROGRAMS -   48

```

Entity Parts List (EN01)

Category: APS-generated report (see *Application Reports*)

Description: The Entity Parts List Report catalogs the components of one or more selected applications, data structures, programs, report mock-ups, screens, subschemas, user macros, APS macros, or COPYLIBs, down to the level of detail that you specify. The report categorizes information based on how it is used in the APS generation process. You can report on:

- The APS entities needed to generate an application—for example, APS macros and entities used internally to generate APS applications.
- Unresolved references that the APS Generator creates—for example, CALL WS-PROG IN WS-PROG. Use this information to help you debug an application.
- The source code that the APS Generator produces. Use this information when you need a code listing.

Together these items provide a record of the complete progression of your application from APS entity definitions to code.

- Comments:**
- Produce the Entity Parts List Report from the Documentation Facility. In addition to the standard types, you can specify non-APS libraries defined at your site.
 - To specify the application that contains the program, report on the global application components, such as data structures and user macros, that are associated with the program in a particular application.
 - To specify the depth of detail that you want to report on, type a value between 1 and 999 in the Explosion Limit field, or leave the field blank to report on all available levels.
 - To specify the kind of data that you want to report on, complete the Use Type field as follows:
 - A(ps) to report on APS product components, such as APS macros
 - I(nfo) to report on unresolved references

- S(source) to report on a component that is not part of the APS product

Example:

```
REPORT CODE: EN01          APS APPLICATION DICTIONARY          PAGE 1
                           ENTITY PARTS LIST                92/07/23 12:18
                           CTSAPS.TEST
```

```
ENTITY TYPE: APSAPPL
ENTITY NAME: *
APPLICATION:
EXPLOSION LIMIT:
USE TYPES: SOURCE
```

```
APSAPPL (DEMOAPPL) 92/07/23 12:17
  APSPROG (DEMOPG1 ) 92/07/23 11:55
    APSDATA (DEMODS1 ) 92/07/23 12:08
    APSSCRN (DEMOSC1 ) 92/07/23 11:57
    DDISYMB (DEMOPSB ) 92/05/15 15:20
      COPYLIB (D2MASTER)
      COPYLIB (D2STOCK )
    USERMACS(DEMOUS1 ) 92/07/23 12:11
    USERMACS(DEMOUS5 ) 92/07/23 12:14
  APSPROG (DEMOPG2 ) 92/07/23 11:56
    APSDATA (DEMODS2 ) 92/07/23 12:08
    APSSCRN (DEMOSC2 ) 92/07/23 12:05
    DDISYMB (DEMOPSB ) 92/05/15 15:20
      COPYLIB (D2MASTER)
      COPYLIB (D2STOCK )
    USERMACS(DEMOUS2 ) 92/07/23 12:13
    USERMACS(DEMOUS6 ) 92/07/23 12:15
  APSPROG (DEMOPG3 ) 92/07/23 11:56
    APSDATA (DEMODS3 ) 92/07/23 12:08
    APSSCRN (DEMOSC3 ) 92/07/23 12:06
    DDISYMB (DEMOPSB ) 92/05/15 15:20
      COPYLIB (D2MASTER)
      COPYLIB (D2STOCK )
    USERMACS(DEMOUS3 ) 92/07/23 12:14
    USERMACS(DEMOUS7 ) 92/07/23 12:15
  APSPROG (DEMOPG4 ) 92/07/23 11:57
    APSDATA (DEMODS4 ) 92/07/23 12:09
    APSSCRN (DEMOSC4 ) 92/07/23 12:07
    DDISYMB (DEMOPSB ) 92/05/15 15:20
      COPYLIB (D2MASTER)
```

```
COPYLIB (D2STOCK )  
USERMACS(DEMOUS4 ) 92/07/23 12:13  
USERMACS(DEMOUS8 ) 92/07/23 12:16
```

1 TARGET WAS LISTED.

Entity Search Utility Report (GS01)

Category: APS-generated report (see *Application Reports*)

Description: The Entity Search Utility Report lets you use search expressions to report on subsets of application data that meet the requirements that you specify. A search expression can be either a literal text string or a regular expression that lets you search for a certain criteria, such as all occurrences of certain data name strings in a group of data structures.

When you generate this report, you specify the level of detail that you want to report on. The available levels depend in part on the type of data that you select for the report.

- Comments:**
- Produce the Entity Search Utility Report from the Documentation Facility.
 - To report on components with common criteria, such as a common name prefix, enter a wildcard as explained below.
 - To report on a program, optionally specify the application that contains it. Do so to report on the global application components, such as data structures and user macros, that are associated with the program in a particular application.
 - To specify the depth of detail that you want to report on, type a value between 1 and 999 in the Explosion Limit field, or leave the field blank to report on all levels.
 - You can specify whether to include APS members in the report.
 - You can specify the search expression that represents the specific information you want to report on.

- A search expression can be a text string or a regular expression. You can create a list of one or more search expressions, and include both text strings and regular expressions.
- A text string is a literal sequence of characters to search on. For example, the string ABC finds ABC. The string is case sensitive. If the string contains spaces, delimit it with single or double quotation marks.
- A regular expression is a pattern of characters. It may include text characters and metacharacters. Metacharacters have special meaning; see below for details.
- Create regular expressions using the following metacharacters.

Metacharacter	Description
.	Match any value you seek. For example, DATA... matches such values as DATA-TY or DATASTR.
" "	Delimit spaces in an expression. For example, "% DEFINE." or '% DEFINE.' match % DEFINE and % DEFINED.
()	In a pair, specify a group that can be a range (such as A-Z) or a list (such as ABCD). Within a group, use only the following metacharacters.
()	Specify the first character in a group.
[]	Specify a range within a group.
\	Represent a metacharacter as itself.
*	Match zero or more occurrences of the single character or character group immediately preceding the *. For example, PROG*1 matches PRO1, PROG1, and PROGGG1 but not PR1, because the characters PRO must be part of the match. The quoted * simply represents itself.
+	Match one or more occurrences of the single character or character group immediately preceding the +. The example is the same as the prior case, but there must be at least one match. The quoted + simply represents itself.

Metacharacter	Description
?	Match zero or one occurrences of the single character or character group immediately preceding this metacharacter. The example is the same as the prior case, but there can be at most one match. The quoted ? simply represents itself.
^	In the first position, represent a logical not, and select data that does not specify the specified criteria. In the first position of an expression, it matches the rest of any expression. In any other position, it represents itself.
\$	Specify the preceding character or group of characters if the metacharacter appears in the last position of the expression and the match appears at the end of the line. For example, WS-CNT\$ matches ADD WS-CNT TO WS-TOT.
\	Indicate that the single metacharacter immediately following the \ is an ordinary text character rather than a metacharacter. For example, [A-Z\^] matches the character range A through Z and the ^ character.

Example:

```
REPORT CODE: GS01                APS APPLICATION DICTIONARY                PAGE      1
                                ENTITY SEARCH UTILITY                    92/07/10  02:06
                                APS.TEST
```

```
APPLICATION:
  ENTITY: USERMACS
  MEMBER: A1UTTREE
EXPLOSION LIMIT:
  USE TYPES:
```

```
SEARCH EXPRESSIONS:
  r:"% *DEFINE "
  r:"% *END "
  R:"% *IF"
  R:"% *ELSE"
```

```
APPL      ENTITY TYPE      ENTITY NAME LINE
.....1.....2.....3.....4.....5.....6.....7.....
      USERMACS      A1UTTREE  1 0019      %DEFINE $TREE-DEFINE(
      3 0030      %IF &TREE = ""
      3 0032      %IF &LENGTH(&TREE) > 6
      3 0036      %IF &DEFINED(&A1UTTREE-<&TREE>-DEFINED)
      3 0041      %IF &POINTER-SIZE = "HALF"
```

Reference

```

4 0044      %ELSE-IF &POINTER-SIZE = "FULL"
4 0047      %ELSE
3 0050      %IF &ALLOC-PARA NOT = ""
3 0053      %IF &DEBUG = 0
4 0055      %ELSE
3 0061      %IF NOT &DEFINED(&ALUTTREE-WORK-AREA)
3 0068      %IF &WS = "LINKAGE"
2 0115      %END
1 0122      %DEFINE $TREE-CLEAR(
3 0126      %IF &TREE = ""
2 0137      %END
1 0147      %DEFINE $TREE-ADD(
3 0158      %IF &TREE = ""
3 0163      %IF &NODEX = ""
4 0165      %ELSE-IF &INDEX( &NODEX, "(" )
3 0168      %IF &PREVX = ""
4 0170      %ELSE-IF &INDEX( &PREVX, "(" )

```

Entity Use Report (EN02)

Category: APS-generated report (see *Application Reports*)

Description: The Entity Use Report lists components that use the target component, as in a COPY or INCLUDE statement. For example, you can get a list of components that use a certain subschema.

When you generate this report, you specify the level of detail that you want to report on. The available levels depend in part on the type of data that you select for the report.

- Comments:**
- Produce the Entity Use Report from the Documentation Facility.
 - If you are reporting on a program, optionally specify the application that contains it. Do so to report on the global application components, such as data structures and user macros, that are associated with the program in a particular application.
 - Specify the depth of detail that you want to report on. To do so, type a value between 1 and 999 in the Explosion Limit field, or leave the field blank to report on all available levels.
 - Specify the kind of data that you want to report on by completing the Use Type field as follows.

- A(ps) to report on APS product components, such as APS macros
- I(nfo) to report on unresolved references not used during APS Generation--for example, CALL WS-PROG-NAME
- S(ource) to report on a component that is not part of the APS product
- Blank to report on all types of information

Example:

```

REPORT CODE: EN02   APS APPLICATION DICTIONARY           PAGE   1
                ENTITY USE REPORT                       02/06/21 14:02
                APS.MVS21DEV
TARGET:  USERMACS(ISPFUSER)
APPLICATION:
EXPLOSION LIMIT:
USE TYPE SELECTIONS:

APSAPPL (APSLINK8)
  APSPROG (A2CNFIG8)
    USERMACS(ISPFMACS)
    USERMACS(ISPFCOMM)
    USERMACS(ISPFUSER)
APSAPPL (APSLINK8)
  APSPROG (A2NDMGD8)
    USERMACS(ISPFMACS)
    USERMACS(ISPFCOMM)
    USERMACS(ISPFUSER)
APSAPPL (APSLINK8)
  APSPROG (A2NDMGU8)
    USERMACS(ISPFMACS)
    USERMACS(ISPFCOMM)
    USERMACS(ISPFUSER)
APSAPPL (APSLINK8)
  APSPROG (A2RECGU8)
    USERMACS(ISPFMACS)
    USERMACS(ISPFCOMM)
    USERMACS(ISPFUSER)
APSAPPL (APSLINK8)
  APSPROG (A2RECV8)
  APSPROG (A2RECGU8)
    USERMACS(ISPFMACS)
    USERMACS(ISPFCOMM)
    USERMACS(ISPFUSER)
APSAPPL (APSLINK8)
  APSPROG (A2RECV8)

```

Reference

```

        USERMACS ( ISPFMACS )
            USERMACS ( ISPFCOMM )
                USERMACS ( ISPFUSER )
    APSAPPL ( APSLINK8 )
        APSPROG ( A2SEND8 )
            USERMACS ( ISPFMACS )
                USERMACS ( ISPFCOMM )
                    USERMACS ( ISPFUSER )
    APSAPPL ( APSLINK8 )
        APSPROG ( A2SENGD8 )
            USERMACS ( ISPFMACS )
                USERMACS ( ISPFCOMM )
                    USERMACS ( ISPFUSER )
    APSAPPL ( APSPC )
        APSPROG ( APSNA )
            USERMACS ( ISPFMACS )
                USERMACS ( ISPFCOMM )
                    USERMACS ( ISPFUSER )

```

ENTRY

Category: S-COBOL structure (see *S-COBOL Structures*)

Purpose Establish an entry point in a COBOL subprogram.

Syntax: `ENTRY literal`
 `.`
 `.`
 `.`
 `[USING identifier1, ..., identifierN]`

- Comments:**
- S-COBOL considers the paragraph where the program enters the subprogram as a main-logic paragraph.
 - Code ENTRY immediately after a paragraph name.
 - Code ENTRY only in a paragraph that is not performed by any other paragraph within the subprogram. At the end of this paragraph, control returns to the calling program, so that you do not need to code EXIT PROGRAM.
 - If the program reaches an EXIT PROGRAM before the end of the paragraph, control returns to the calling program.

Error Handling

CICS

Description: Test for any Exceptional condition in the CICS environment.

APS generates a CICS IGNORE CONDITION command that ignores all CICS Exceptional conditions, and generates an 88-level EIBRCODE structure.

The APS/CICS default for inline error checking is to generate a global CICS IGNORE condition. You can generate a NOHANDLE on a call-by-call basis. Flags for both are in the APS CNTL file APCICSIN.

Example: Test for the MAPFAIL condition and perform a user-defined paragraph to handle it.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
NTRY
      IF MAPFAIL
          PERFORM MAPFAIL-PARA
```

EIBRCODE Structure:

The variable that controls generation of the EIBRCODE structure is &CIC-APS-EIBRCODE, which resides in APCICSIN. The following is the APS-generated EIBRCODE structure, which resides in APCICSTP.

```
01  APS-EIBFN-EIBRCODE .
    05  APS-EIBFN                PIC X(01) .
    05  APS-EIBRCODE             PIC X(06) .

01  FILLER                       REDEFINES
    APS-EIBFN-EIBRCODE .
    05  APS-EIBFN-EIBRCODE-X     PIC S9(04) COMP .
    88  CBIDERR                  VALUE +1259 .
    88  DISABLED                 VALUE +1549 .
    88  DSIDERR                  VALUE +1537 .
    88  DSSTAT                   VALUE +7684 .
    88  DUPKEY                   VALUE +1668 .
    88  DUPREC                   VALUE +1666 .
    88  ENDDATA                  VALUE +4097 .
    88  ENDFILE                  VALUE +1551 .
    88  ENDINPT                  VALUE +1218 .
    88  ENQBUSY                  VALUE +4658 .
```

88	ENVDEFERR	VALUE +4329.
88	EODS	VALUE +1040.
88	EOF	VALUE +1028 +1217.
88	EXPIRED	VALUE +4128.
88	FUNCERR	VALUE +7688.
88	IGREQCD	VALUE +1258.
88	ILLOGIC	VALUE +1538.
88	INVERRTERM	VALUE +6176.
88	INVMP SZ	VALUE +6152.
88	INVREQ	VALUE +736 +1248 +1544 +2592 +3808 +4351 +4832 +5122 +6145 +6880.
88	INVTSREQ	VALUE +4116.
88	IOERR	VALUE +1664 +2052 +2564 +4100 +5127.
88	ISCINVREQ	VALUE +1745 +2257 +2769 +4305.
88	ITEMERR	VALUE +2561.
88	JIDERR	VALUE +5121.
88	LENGERR	VALUE +1249 +1761 +2273 +2785 +3297 +4321 +5126 +6369 +7905.
88	MAPFAIL	VALUE +6148.
88	NOJBUFSP	VALUE +5129.
88	NOPASSBKRD	VALUE +1255.
88	NOPASSBKWR	VALUE +1256.
88	NOSPACE	VALUE +1667 +2064 +2568.

88	NOSTG	VALUE	+3298.
88	NOTALLOC	VALUE	+1237.
88	NOTFND	VALUE	+1665
			+4225.
88	NOTOPEN	VALUE	+1548
			+2056
			+5125.
88	PGMIDERR	VALUE	+3585.
88	QBUSY	VALUE	+2240.
88	QIDERR	VALUE	+2050
			+2562.
88	QZERO	VALUE	+2049.
88	RDATT	VALUE	+1252
			+6372.
88	RETPAGE	VALUE	+6146.
88	RTEFAIL	VALUE	+6272.
88	RTESOME	VALUE	+6208.
88	SEgidERR	VALUE	+1540.
88	SELNERR	VALUE	+7692.
88	SESSBUSY	VALUE	+1236.
88	SESSIONERR	VALUE	+1234.
88	SIGNAL	VALUE	+1253.
88	SYSBUSY	VALUE	+1235.
88	SYSIDERR	VALUE	+1232
			+1744
			+2256
			+2768
			+4304.
88	TERMIDERR	VALUE	+1254
			+4114.
88	TRANSIDERR	VALUE	+4113.
88	UNEXPIN	VALUE	+7696.
88	WRBRK	VALUE	+1251
			+6371.
88	ERROR-FOUND	VALUE	+1259
	+1537	+7684	+1668 +1666 +4097 +1551
	+1218	+4658	+4329 +1040 +1028 +1217
	+4128	+7688	+1258 +1538 +6176 +6152
	+736	+1248	+1544 +2592 +3808 +4351
	+4832	+5122	+6145 +6880 +4116 +1664
	+2052	+2564	+4100 +5127 +1745 +2257
	+2769	+4305	+2561 +5121 +1249 +1761
	+2273	+2785	+3297 +4321 +5126 +6369
	+7905	+6148	+5129 +1255 +1256 +1667
	+2064	+2568	+3298 +1237 +1665 +4225
	+1548	+2056	+5125 +3585 +2240 +2050
	+2562	+2049	+1252 +6372 +6146 +6272
	+6208	+1540	+7692 +1236 +1234 +1253

```

+1235 +1232 +1744 +2256 +2768 +4304
+1254 +4114 +4113 +7696 +1251 +6371.
05 FILLER PIC X(05).

```

IDMS DB

Description: Test for any error condition in the IDMS DB environment. Check the call status in your program with flags provided by the APS/IDMS DB Generator. All flags are COBOL 88-level condition names.

Flag	Status Code	Error Condition
AB-ON-REC	0001 thru 0306 0308 thru 0325 0327 thru 0625 1206 thru 9999	Any error other than those listed
DUP-ON-REC	0705 0805 1205	Duplicate key
END-ON-REC	0307	End of set, area, or index
NTF-ON-REC	0326 0626	Record not found
OK-ON-REC	0000	Successful operation
POS-ON-REC	All values of AB-ON-REC whose last 2 bytes are 06, 13	Positioning error
VIO-ON-REC	All values of AB-ON-REC whose last 2 bytes are 01, 02, 08, 09, 10, 14, 15, 23, 31	Update violates IDMS DB rules

After any APS/IDMS call, the value of ERROR-STATUS moves to DBIO-STATUS. The program checks the associated 88-level flags for status checking. If a DB call contains a record name, such as DB-STORE REC CUSTOMER, you can move the value of ERROR-STATUS to record-STATUS, as well as DBIO-STATUS, and the program checks both fields.

Generated record-STATUS flags have the following format.

```

ORDER-STATUS PIC X(04) VALUE '0000'.
88 STABLE-ON-ORDER VALUE '0000'.
88 AT-END-ON-ORDER VALUE '0307'.
88 INVALID-KEY-ON-ORDER VALUE '0326' '0626'.
88 INVALID-DUP-ON-ORDER VALUE '1205' '0805' '0705'.
88 ABNORMAL-ON-OR VALUE '0001' THRU '0306'
'0308' THRU '0325'
'0327' THRU '0625'

```

```

'0627' THRU '0704'
'0706' THRU '0804'
'0806' THRU '1204'
'1206' THRU '9999'.
```

```

DB-OBTAIN REC IDMSREC AREA CUSTOMER-REGION FIRST
IF OK-ON-REC
  DB-OBTAIN REF CUSTOMER REC ORDER FIRST
  IF OK-ON-ORDER
    PERFORM ORDER-PROCESSING-PARA
  ELSE-IF AT-END-ON-ORDER
    ... OR INVALID-KEY-ON-ORDER
    PERFORM NO-ORDERS-PARA
  ELSE
    PERFORM ABNORMAL-ON-ORDER-PARA
```

DB-OBTAIN and DB-GET generate record-STATUS flag values (when the record name is given). All other calls generate DBIO-STATUS values.

Example:

```

01 DBIO-STATUS PIC X(04) VALUE '0000'.
   88 OK-ON-REC VALUE '0000'.
   88 END-ON-REC VALUE '0307'.
   88 NTF-ON-REC VALUE '0326' '0626'.
   88 DUP-ON-REC VALUE '1205' '0805'
      '0705'.
   88 AB-ON-REC VALUE '0001' THRU '0306'
      '0308' THRU '0325'
      '0327' THRU '0625'
      '0627' THRU '0704'
      '0706' THRU '0804'
      '0806' THRU '1204'
      '1206' THRU '9999'.
01 APS17-STATUS REDEFINES DBIO-STATUS.
   02 APS17-MAJOR-CODE PIC X(02).
   02 APS17-MINOR-CODE PIC X(02).
      88 POS-ON-REC VALUE '06' '13'.
      88 VIO-ON-REC VALUE '01' '02'
          '08' '09'
          '10' '14'
          '15' '23'
          '31'.
```

IMS DB

Description: Test for any error condition in the IMS DB environment. Check the call status in your program with flags provided by the APS/IMS DB Generator. All flags are COBOL 88-level condition names.

Flag	Status Code	Error Condition
AB-ON-REC	Any not listed below	For any error code not listed below
DUP-ON-REC	I, NI LB	Call failed because the new segment would create a duplicate for a key or sequence field defined as unique
END-ON-REC	GB	End of database reached
NTF-ON-REC	GE GB	Requested record not found
OK-ON-REC	2 spaces, GA GD GK	Everything is OK
POS-ON-REC	DJ LC LD LE	Positioning error; requested positioning not established
RTY-ON-REC	GG	Record not available; retry
VIO-ON-REC	AM DA DX RX IX	Update violates IMS DB rules

Abnormal Error Processing:

By default, the APS-supplied IMS database error macros--\$IM-ERR-CONDITION and \$IM-ERR-ACTION--use APS-supplied status flags.

The macros also call DFS0AER, the IMS-supplied error display routine. To enable the call to DFS0AER, go to the APS CNTL file APSDBDC and set the variable % &IM-USE-DFS0AER to 1. If you don't have DFS0AER at your installation, or you want to disable it, just leave the variable set to 0.

APS supplies two error macros in the APSMACS file IMSPHYS.

\$IM-ERR-CONDITION Specifies the conditions for which the database return status indicates an error. The conditional statement IF AB-ON-REC tests for the APS data base status flags (found in the generated Working-Storage field IM-FLGS), and the IMS status codes (in the field IM-STATUS).

\$IM-ERR-ACTION Contains the procedures executed when the condition specified in **\$IM-ERR-CONDITION** is True. When AB-ON-REC codes are returned, this macro calls DFS0AER, the IMS-supplied error display routine. Reminder: To enable calling DFS0AER, go to APSTBDC and make sure **% &IM-USE-DFS0AER** is set to 1.

To modify **\$IM-ERR-CONDITION**, write an overriding macro of the same name in the USERMACS macro library. Use the override macro for

- A specific application, enter the macro name in the Application Painter field USERMACS, and specify in the Loc(ation) field a location after the Identification Division but before the Procedure Division.
- An entire Project and Group, code in your project.group.APSPROJ file the APS customization exit name **\$DB-SUBSCHEMA-EXIT-2**, and an **% INCLUDE** statement that includes the override macro. For example:

```
% DEFINE $DB-SUBSCHEMA-EXIT-2
    % INCLUDE USERMACS(MY-OVERRIDE-MAC)
% END
```

Notes:

- The override macro for **\$IM-ERR-CONDITION** must generate one simple or compound S-COBOL conditional statement that tests APS data base status flags (found in the generated Working-Storage field IM-FLGS), the IMS status code (in the field IM-STATUS), or both.
- To modify **\$IM-ERR-ACTION**, write an overriding macro of the same name. You can use the override macro for a specific application or an entire Project and Group, as detailed above.
- The override macro for **\$IM-ERR-ACTION** must generate S-COBOL procedural code for the action specified when a condition(s) tested by **\$IM-ERR-CONDITION** is True. You can use the following

parameters, whose values are passed to \$IM-ERR-ACTION after a bad database call.

Parameter	Description
&ERR-PCB	Name of PCB used for call.
&ERR-MACNAME	Name of call resulting in error.
&ERR-FUNC	IMS function code used in call.
&ERR-SEGNAME	Name of segment requested in call.
&ERR-PAR-SEGNAME	Segment name of parent requested in call.
&ERR-IOAREA	COBOL name of record I/O area.
&ERR-PAR-IOAREA	COBOL name of parent record I/O area.
&ERR-USER-MSG	COBOL name of 72 byte error message field.
&ERR-SSA1 thru 15	SSA(s) used in call.
&IM-LVL-MAX	Maximum level of the call.

- Examples:**
- Receive the status flag NTF-ON-REC on a DB-OBTAIN with a single segment level to indicate the requested employee number does not exist.

```
DB-OBTAIN RECORD EMPLOYEE-MASTER
... WHERE EMP-NO = NEEDED-EMP-NO
```

- In contrast, receiving NTF-ON-REC on a DB-OBTAIN requesting the return of segments at three levels, cannot specify the segment at which the call failed. Resolve this by checking the IMS error fields.

```
DB-OBTAIN RECORD EMPLOYEE-MASTER
... WHERE EMP-NO = NEEDED-EMP-NO REC WEEKLY-TIME-SEG
... WHERE WEEK-END-DATA = PERIOD-DATE REC PROJECT-TIME-REC
... WHERE PROJ-CODE = CURRENT-PROJ
```

IMS Error Fields

IMS provides error fields that show how far your call was processed prior to failure.

IM-DB-PCB-SEGLEV	Lowest level of the segment found in the database, for example, 15, if a 15th-level segment is found. Default 00.
IM-DB-PCB-SEGNAME	8-character IMS name for the lowest-level segment located.

IM-DB-PCB-KEY-FEED-BACK	Concatenated key information for the path from the root-level to the lowest-level segment found.
IM-DB-PCB-KEY-KFBLEN	Length of data in the IM-DB-PCB-KEY-FEED-BACK field.

IMS DC

Description: Test for any error condition in the IMS DC environment. Check the call status in your program with flags provided by the APS/IMS DC Generator. All flags are COBOL 88-level condition names.

Flag	Status Code	Error Condition
AB-ON-DC-CALL	CH X1 X8	Category 5 status code return; call not complete.
FP-ERR	<i>FF FH FS FV</i>	Category 3 status code return. Fast Path error; call complete.
NO-MORE-MSGS	QC	Category 3 status code return on the TP call; no more input messages exist.
NO-MORE-SEGS	QD	Category 3 status code return; no more segments exist for this message.
SEG-NOT-FOUND	GE	Category 1 status code return; segment not found.
OK-ON-DC-CALL	2 spaces, CC CE CF CG CI CJ CK CL FD FW FF FH FS FV GE QC QD	Categories 1 and 2 status code return; processing proceeds.
SEC-VIO	A4 FI	Category 4 status code returned; security violation occurred; call not complete.
SPA-IO-ERR	XA XB XE XF XG X1 X2 X3 X4 X5 X6 X7 X8 X9	Categories 4 and 5 status code return; SPA error; call not complete.

Flag	Status Code	Error Condition
TP-PGM-ERR	AA AB AD AL AP AT AY AZ A1 A2 A3 A4 A5 A6 A7 A8 A9 CA CB CD QE QH	Category 4 status code return; programming error; call not complete.

Abnormal Error Processing

By default, the APS-supplied IMS data communication error macros--\$TP-ERR-CONDITION and \$TP-ERR-ACTION--use APS-supplied status flags.

The macros also call DFS0AER, the IMS-supplied error display routine. To enable the call to DFS0AER, to to the APS CNTL file APSDBDC and set the variable % &IM-USE-DFS0AER to 1. If you don't have DFS0AER at your installation, or you want to disable it, just leave the variable set to 0.

APS supplies these two error macros in the APSMACS file IMSPHYS.

\$TP-ERR-CONDITION	Specifies the conditions for which the data communication return status indicates an error. The conditional statement IF AB-ON-DC-CALL tests for the APS data communication flags (found in the generated Working-Storage field TP-FLGS), and the IMS status codes (in the field TP-STATUS).
\$IM-ERR-ACTION	Contains the procedures executed when the condition specified in \$TP-ERR-CONDITION is True. When AB-ON-DC-CALL codes are returned, this macro calls DFS0AER, the IMS-supplied error display routine.

Note: To enable calling DFS0AER, go to APSDBDC and make sure % &IM-USE-DFS0AER is set to 1.

To modify \$TP-ERR-CONDITION, write an overriding macro of the same name in the USERMACS macro library. You can use the override macro for a specific application or an entire Project and Group.

The override the macro for \$TP-ERR-CONDITION, generate a simple or compound S-COBOL conditional statement that tests APS data communication status flags (found in the generated Working-Storage field TP-FLGS), the IMS status code (in the field TP-STATUS), or both.

To modify \$TP-ERR-ACTION, write an overriding macro of the same name. You can use the override macro for a specific application or an entire Project and Group. The override macro must generate S-COBOL procedural code for the action specified when a condition(s) tested by \$TP-ERR-CONDITION is True. You can use the following parameters that are passed to \$TP-ERR-ACTION after a bad data communication call.

Parameter	Description
&TP-ERR-PCB	Name of the I/O PCB used for the call
&TP-ERR-FUNC	Generated Working Storage field, IM-CALL-FUNC, to which the DC call function is moved prior to the call
&TP-ERR-MSG	Generated 72-byte Working Storage field, IM-ERR-MSG, containing an error message
&TP-ERR-SEG-IOAREA	Generated Working Storage field, either TP-SEGMENT or user record area
&TP-ERR-SPA	Generated Working Storage field, TP-SPA, present only in IMS conversational programs

SQL

Description: Test for any error condition in the SQL environment. Check the call status in your program with flags provided by the APS/SQL Generator; test SQLCODE. All flags are COBOL 88-level condition names.

Flag	Error Condition
AB-ON-REC	Any error not listed in this table.
DB2-DEADLOCK	DB-PROCESS calls check this status to ensure the cursor is not already closed before closing it; SQL closes the cursor if database is locked.
DUP-ON-REC	DB-STORE failed because the row already exists; duplicates not allowed.
END-ON-REC	End of table or cursor set reached.
NTF-ON-REC	Requested row not found.
OK-ON-REC	Operation successful.

Flag	Error Condition
RI-ON-REC	Referential Integrity check successful (corresponds to SQLCODE -532 to -530).

Abnormal Error Processing

APS/SQL provides flags for you to code in your program to check the status of SQL calls. When the AB-ON-REC flag is returned, it invokes an abnormal condition processing macro (\$D2-ERROR-PARA) that displays a message and, for IMS and CICS programs, terminates the program.

If your DC target is IMS, and you are not running IMS under BTS, set &D2-EXEC-UNDER-BTS=NO in the APS CNTL file APDB2IN. This eliminates "Display" statements in the APS-generated error handling routine.

To disable &D2-ERROR-PARA, go to the APS CNTL file APDB2IN and set &D2-AUTO-ERROR-HANDLING to OFF.

You can modify error processing in two ways.

- Modify the APS error processing paragraph &D2-AUTO-ERROR-HANDLING, which resides in the APS CNTL file APDB2IN macro \$DB2-ERROR-SETUP
- Modify which status codes should be considered error conditions, in the APS CNTL file APDB2IN macro \$DB2-CHECK-RETURNS-AUX. In addition, you can change the status of a referential integrity constraint from an abnormal condition, to an invalid key. Go to the APS CNTL file APDB2IN and set the flag &D2-RI-IS-INVALID-KEY to Yes.

Trace Flag

Description: Use the S-COBOL trace flag for debugging. This facility displays where APS performs each paragraph.

The Trace facility differs from the IBM READY-TRACE because it displays the paragraph name only when a PERFORM executes a paragraph, and not each time a loop executes.

Note: To activate the Trace facility, specify SCBTRACE on the Precompiler Options screen.

Syntax: WORKING-STORAGE SECTION.
 .
 .
 .
 02 SAGE-TRACE-FLAG PIC X VALUE "T".
 .
 .
 .
 PROCEDURE DIVISION.
 .
 .
 .
paragraphname
 IF SAGE-TRACE-FLAG = TRUE
 DISPLAY "EXEC:--*paragraphname*--".
 .
 .
 .

- Comments:**
- You can incorporate logic to set SAGE-TRACE-FLAG to FALSE until some selected point in the program, and then set it to TRUE.
 - To turn the TRACE feature off at run-time, code:

```
MOVE FALSE TO SAGE-TRACE-FLAG
```

VSAM Batch

Description: Test for any error condition in the VSAM batch environment. Check the call status in your program with flags provided by the APS/VSAM Batch Generator. All flags are COBOL 88-level condition names.

Flag	Status Code	Error Condition
OK-ON-REC	00	Successful operation
DUP-ON-REC	02	Duplicate key; duplicates allowed
END-ON-REC	10	End of file
INV-ON-REC	20 21 22 23 24	Invalid key condition
IVD-ON-REC	22	Duplicate key; duplicates not allowed
NTF-ON-REC	23	Record not found
AB-ON-REC	30 34 90 91 92 93 94 95 96 97	Abnormal condition

Example:

```
DB-STORE REC CUST-RECORD
IF OK-ON-REC
  SCREEN-MSG = 'CUSTOMER ADDED TO FILE'
```

```
ELSE-IF IVD-ON-REC
  SCREEN-MSG = 'DUPLICATE CUSTOMER - NOT ADDED'
```

Abnormal Error Processing

When the AB-ON-REC flag is returned, it invokes an abnormal condition processing macro (\$DB-ERR-CALL) to identify and process run-time I/O errors, terminate the program, and display a message giving the following information.

- DB call in error
- Native VSAM Batch call name
- File name
- Program name
- Status code
- Program termination method; default is a COBOL STOP RUN

Note: APS identifies the status code values listed in the previous table; the developer should test for all other conditions, such as, 00, 02, 10, 20, 21, 22, 23, and 24.

Program generation options

- Exclude specific status code values from AB-ON-REC.
- Deactivate the APS/VSAM Batch abnormal condition processing routine.

There are three ways to modify AB-ON-REC processing.

- Exclude certain CICS Exceptional Conditions and ISI-Errors, or batch conditions, from being AB-ON-REC conditions. To do so, go to the APS CNTL file APVSAMIN and override their variables.
- Disable \$DB-ERR-CALL. In the APVSAMIN file, set &VS-AUTO-ERROR-HANDLING to No.
- Override \$DB-ERR-CALL.

To override \$DB-ERR-CALL, define (or % INCLUDE in your program, using the SYM1 keyword) your own \$DB-ERR-CALL macro.

```

Example:      % DEFINE $DB-ERR-CALL
                  PERFORM LOG-VSAM-ERROR
                  % END
                  % SET EPILOGUE $LOG-VSAM-ERROR
                  % DEFINE $LOG-VSAM-ERROR
                    % SET WORKING-STORAGE
                    COPY LOGDATA.
                  % SET PROCEDURE
                    LOG-VSAM-ERROR.
                    /* CAPTURE EIB DATA
                    MOVE EIBFN      TO CA-EIBFN
                    MOVE EIBRCODE TO CA-EIBRCODE
                    MOVE EIBDS     TO CA-EIBDS
                    MOVE EIBDATE  TO CA-EIBDATE
                    MOVE EIBTIME  TO CA-EIBTIME
                    MOVE EIBTASKN TO CA-EIBTASKN
                    MOVE EIBTRMID TO CA-EIBTRMID
                    MOVE EIBTRNID TO CA-EIBTRNID
                    /* TRANSFER CONTROL TO LOG PROGRAM
                    CICS XCTL
                    ... PROGRAM('LOGERROR')
                    ... COMMAREA(CA-EIB-AREA)
                    ... LENGTH(CA-EIB-AREA-LENGTH)

                  % END

```

VSAM Online

Description: Test for any error condition in the VSAM online environment. Check the call status in your program with flags provided by the APS/VSAM Generator. All flags are COBOL 88-level condition names. Two equivalent sets of flags, APS/CICS VSAM and APS/CICS EIBRCODE, are provided.

VSAM Flag	EIBRCODE Flag	ISI-Errors/ Exceptional Condition	Error Condition
AB-ON-REC	DSIDERR ILLOGIC IOERR LENGERR NOSPACE NOTOPEN SYSIDERR	DSIDERR ILLOGIC IOERR LENGERR NOSPACE NOTOPEN SYSIDERR	Abnormal condition
DUP-ON-REC	DUPKEY	DUPKEY	Duplicate key; duplicates allowed
END-ON-REC	ENDFILE	ENDFILE	End of file
INV-ON-REC	NOTFND DUPREC	NOTFND DUPREC	Invalid key condition
IRQ-ON-REC	INVREQ	INVREQ	Invalid request

VSAM Flag	EIBRCODE Flag	ISI-Errors/ Exceptional Condition	Error Condition
IVD-ON-REC	DUPREC	DUPREC	Duplicate key; duplicates not allowed
NTF-ON-REC	NOTFND	NOTFND	Record not found
OK-ON-REC	N/A	N/A	Successful operation

Examples: With APS/CICS VSAM flags:

```
DB-STORE REC CUST-RECORD
IF OK-ON-REC
    SCREEN-MSG = 'CUSTOMER ADDED TO FILE'
ELSE-IF IVD-ON-REC
    SCREEN-MSG = 'DUPLICATE CUSTOMER - NOT ADDED'
```

With APS/CICS EIBRCODE flags:

```
DB-STORE REC CUST-RECORD
IF EIBRCODE = LOW-VALUES
    SCREEN-MSG = 'CUSTOMER ADDED'
ELSE-IF DUPREC
    SCREEN-MSG = 'DUPLICATE CUSTOMER - NOT ADDED'
```

Abnormal Error Processing

When the AB-ON-REC flag is returned, it invokes an abnormal condition processing macro (\$D2-ERR-CALL) to identify and process run-time I/O errors, terminate the program, and display a message giving the following information.

- DB call in error
- Native VSAM Batch call name
- File name
- Program name
- Status code
- Program termination method, which is a generated TERM
- For a LENGERR condition, the actual record length provided by CICS (derived from APS-*shortreclname*-VAR after CICS updates this field)

Note: APS identifies the status code values listed in the previous table; all other conditions, such as, DUPKEY, DUPREC, ENDFILE, INVREQ, and NOTFND, are processed by the programmer.

Program generation options:

- Exclude specific CICS exceptional conditions and ISI-ERRORS from AB-ON-REC.
- Deactivate the APS abnormal condition processing routine.

There are three ways to modify AB-ON-REC processing.

- Exclude certain CICS Exceptional Conditions and ISI-Errors, or batch conditions, from being AB-ON-REC conditions. To do so, go to the APS CNTL file APVSAMIN and override their variables.
- Disable \$DB-ERR-CALL. In the APVSAMIN file, set &VS-AUTO-ERROR-HANDLING to No.
- Override \$DB-ERR-CALL.

To override \$DB-ERR-CALL, define (or % INCLUDE in your program, using the SYM1 keyword) your own \$DB-ERR-CALL macro.

Example:

```
% DEFINE $DB-ERR-CALL
    PERFORM LOG-VSAM-ERROR
% END
% SET EPILOGUE $LOG-VSAM-ERROR
% DEFINE $LOG-VSAM-ERROR
    % SET WORKING-STORAGE
    COPY LOGDATA.
    % SET PROCEDURE
    LOG-VSAM-ERROR.
    /* CAPTURE EIB DATA
    MOVE EIBFN      TO CA-EIBFN
    MOVE EIBRCODE TO CA-EIBRCODE
    MOVE EIBDS      TO CA-EIBDS
    MOVE EIBDATE   TO CA-EIBDATE
    MOVE EIBTIME   TO CA-EIBTIME
    MOVE EIBTASKN  TO CA-EIBTASKN
    MOVE EIBTRMID  TO CA-EIBTRMID
    MOVE EIBTRNID  TO CA-EIBTRNID
    /* TRANSFER CONTROL TO LOG PROGRAM
    CICS XCTL
    ... PROGRAM('LOGERROR')
```

```

... COMMAREA(CA-EIB-AREA)
... LENGTH(CA-EIB-AREA-LENGTH)
% END

```

Error Processing Messages

Category: Screen Painter feature (see *Field Edits*)

Description: Code default error messages to display when the end user enters invalid data or neglects to enter data that is required. You can code these messages for each screen field, or globally for all screen fields.

Note: A field-specific message overrides a global message.

Procedure: *Field-Specific*

To assign an error message for a specific field, follow these steps.

- 1 From the Screen Painter, access the Field Edit Facility.
- 2 Access the Error Processing screen by selecting the Error Processing prompt on any Field Edit screen.
- 3 Code messages of up to 75 characters. This message overrides any global default messages you entered on the Parm screen for screen fields.
- 4 Specify the attribute values for fields that fail input edits using the following syntax:

```
attribute+attribute+...
```

The default attributes are bright intensity and cursor positioning on the field.

- 5 You can copy the default messages and attributes from the Parm screen, and apply them to the current field. To do so, enter **def** or **defaults** in the **Command** field. You can the modify the message as desired.

Global

To assign an error message that applies to all screen fields, or to bypass input edits for the screen, follow these steps.

- 1 From the Screen Painter, access the Field Edit Facility.
- 2 From the Field Selection screen, enter **pm** or **parm** in the **Command** field. The Parm Screen screen displays.
- 3 Enter the name of the field on your screen that will display the global error message.
- 4 Enter the text to display when the data does not pass field edits and enter the text to display when required data is not entered in the appropriate fields.
- 5 Specify the attribute values for fields that fail input edits using the following syntax:

attribute+attribute+...

The default attributes are bright intensity and cursor positioning on the field.

- 6 To define conditions for bypassing input edits for the screen, select Bypass Edits on the Parm Screen screen. A subsequent Parm Screen screen for bypassing edits displays. You can define bypass conditions for one field per screen; if any of these conditions occur, APS bypasses field edits for the entire screen. If the field is in a repeated block, APS bypasses edits for all fields in that row occurrence only.
- 7 Complete the fields on this screen as follows.

Field	Description and Values
Field Name	Specify any field on the screen, including a field in a repeated block, to bypass.
Value(s)	Specify the value or values that let end users bypass input edits. Valid COBOL reserved words are spaces, low-values, and high-values.
Additional Value(s)	Enter as many additional bypass values that can fit on the line; separate each value with a comma.
Program Function Keys	Select the PF keys the end user can press to bypass the input edits.

ESCAPE

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Exit from the current paragraph.

Syntax: ESCAPE

Comments: Processing resumes with the first statement after the statement performing the paragraph, except if ESCAPE occurs in the first or main logic paragraph of

- The main program--control returns to the operating system by generating an EXIT PROGRAM statement.
- A called program--control passes to the first statement after the CALL statement in the calling program. The called program generates an EXIT PROGRAM statement.

Example: If the condition in line 2220 is true, pass control back to line 2050.

```
.  
. .  
002030 CHECK-DATA  
002040     PERFORM STEP-1  
002050     ADD 1 TO COUNTER  
. .  
002210 STEP-1  
002220     IF X = 1  
002230         ESCAPE  
002240     ADD 1 TO FIELD-A
```

EVALUATE

Category: S-COBOL structure (see *S-COBOL Structures*)

Purpose Perform logic by cases, such as a decision table.

Syntax: Format 1, standard evaluation procedure:

```
EVALUATE identifier
WHEN valuexpression1
    statementblock
[
    .
    .
    .
WHEN valuexpressionN
    statementblockN ]
[WHEN OTHER
    statementblock ]
```

Format 2, decision table:

```
EVALUATE identifier1[, ..., identifierN]
WHEN valuexpression1[, ..., valuexpressionN]
    statementblock
[
    .
    .
    .
WHEN valuexpressionN+1[, ..., valuexpressionN+N]
    statementblock ]
[WHEN OTHER
    statementblock ]
```

Parameters: *identifier* Any COBOL identifier

valuexpression In Format 1, a data name or a group of COBOL literals, identifiers, and arithmetic expressions forming *expression1*, *expression2*, . . .

In Format 2, one of the following
 ANY, or
expression1 [THRU *expression2*]
 ... OR *expression3*[THRU *expression4*]
 ... [OR *expression5* [THRU *expression6*]]

Symbols such as =, <, and > are not valid.

- Comments:**
- For run time efficiency, APS generates the GO TO ... DEPENDING ON translation with Format 1, if the code meets the following criteria.
 - Each WHEN valuexpression is a numeric literal.
 - The literals increase in value from left to right, top to bottom.
 - The largest (last) literal in the statement is less than four times as big as the number of WHEN clauses, not counting WHEN OTHER.

If the above criteria are met and the evaluated field is not defined as numeric, a COBOL error occurs.

- Only the first WHEN condition met is executed.
- If a WHEN condition is true and it does not include an optional statement block, program control passes to the next statement with the same or less indentation as the word EVALUATE.
- To prevent logic from falling through if no WHEN conditions are met, code WHEN OTHER.
- EVALUATE does not evaluate 88-levels, as the COBOL/2 EVALUATE statement does.
- If a COBOL/2 EVALUATE statement is also valid S-COBOL EVALUATE syntax, APS processes it as S-COBOL statement.
- You can evaluate a maximum of 255 conditions/fields. You can code a maximum of 102 WHEN conditions.
- You can use EVALUATE to code the NEXT SENTENCE concept of passing program control out of a particular construct to the next executable statement.

Example: Create a mailing list that includes new subscribers (less than 1 year) and preferred subscribers (more than 5 years) broken down by region.

```
EVALUATE MONTHS, REGION
WHEN 1 THRU 11, 'EAST'
    WRITE NEW-EAST-REC
WHEN 1 THRU 11, 'WEST'
    WRITE NEW-WEST-REC
WHEN 61 THRU 9999, 'EAST'
    WRITE PREFERRED-EAST-REC
WHEN 61 THRU 9999, 'WEST'
    WRITE PREFERRED-WEST-REC
```

Exit Points

Category: Database calls (see *Database Calls*)

Compatibility: SQL, VSAM Batch, and VSAM Online targets

Description: At various program locations, you can write your own rules to customize methods. In your program, you write a rule using one of the APS-supplied predefined rule names, and APS invokes it at its proper location. You can write rules that are automatically invoked at the beginning and end of the processing using either of these predefined rule name formats.

SQL

For DB-ERASE:

```
$D2-DB-ERASE-BEGIN-EXIT  
$D2-DB-ERASE-END-EXIT
```

For DB-MODIFY:

```
$D2-DB-MODIFY-BEGIN-EXIT  
$D2-DB-MODIFY-END-EXIT
```

For DB-OBTAIN:

```
$D2-DB-OBTAIN-BEGIN-EXIT  
$D2-DB-OBTAIN-END-EXIT
```

For DB-PROCESS:

```
$D2-DB-PROCESS-BEGIN-EXIT  
$D2-DB-PROCESS-END-EXIT
```

For DB-STORE:

```
$D2-DB-STORE-BEGIN-EXIT  
$D2-DB-STORE-END-EXIT
```

VSAM Batch and VSAM Online

For DB-ERASE:

```
$VS-DB-ERASE-BEGIN-EXIT  
$VS-DB-ERASE-END-EXIT
```

For DB-MODIFY:

```
$VS-DB-MODIFY-BEGIN-EXIT
$VS-DB-Modify-END-EXIT
```

For DB-OBTAIN:

```
$VS-DB-OBTAIN-BEGIN-EXIT
$VS-DB-OBTAIN-END-EXIT
```

For DB-PROCESS:

```
$VS-DB-PROCESS-BEGIN-EXIT
$VS-DB-PROCESS-END-EXIT
```

For DB-STORE:

```
$VS-DB-STORE-BEGIN-EXIT
$VS-DB-STORE-END-EXIT
```

EXIT PROGRAM

Category: S-COBOL structure (see *S-COBOL Structures*)

Purpose End the execution sequence of a program or subprogram and return control to the invoking source.

Syntax: EXIT PROGRAM

- Comments:**
- EXIT PROGRAM can appear anywhere in an S-COBOL paragraph.
 - In a called program, EXIT PROGRAM passes control to the first statement after the CALL statement in the calling program.
 - An EXIT PROGRAM terminates all APS programs, unless you include a STOP RUN or GOBACK (IBM COBOL extension).
 - APS generates an EXIT PROGRAM at the end of the first paragraph of any called program so that you do not need to code it.

Expressions, SQL

Compatibility: SQL target

Description: APS/SQL supports expressions for the DB-DECLARE, DB-OBTAIN, and DB-PROCESS calls.

Sample Expressions	Return INTO Host	Null Indicator
INTEGER(DOB) - INTEGER(PERF - 1) (WS-RET-INT)	WS-RET-INT	IND-DOB
CURRENT DATE - 1 DAY (WS-CURRDATE)	WS-CURRDATE	-----
RATE + (DED / 2.0)	RATE	IND-RATE
YTD + RATE + DED / 2	YTD	IND-YTD
PERF / 2	PERF	IND-PERF
INTEGER(YTD - DED) * INTEGER(RATE) (WS-INT-FLD)	WS-INT-FLD	INT-YTD
HIREDTIME + YEAR(CURRENT DATE) YEARS	HIREDTIME	IND-HIREDTIME
SUM(YTD) / 2	YTD	IND-YTD
AVG(DED - 1) / 5 (WS-AVG-DED)	WS-AVG-DED	IND-DED
SUM(RATE) + SUM(DED) (WS-RATE-DED)	WS-RATE-DED	IND-RATE

Comment: If an INTO variable is not specified, the result returns to the host variable for the first column.

Examples:

```

DB-OBTAIN REC TABLE1
... RATE + (DED / 2.0)
... PERF / 2
... INTEGER(DOB) - INTEGER(PERF - 1) (WS-RET-INT)
... WHERE PERF > 100

DB-OBTAIN REC RECORD1-RED
... RATE + YEAR_TO_DATE (WS-CALC-RATE)
... MONTH(HIREDATE) - DAY(HIREDATE) (WS-DATE-RETURN
... AVG(RATE - 1) / 4 (WS-AVG-RATE)
... YEAR((CURRENT DATE - HIREDTIME), WS-CALC-DATE, Y)
... CHAR((HIREDTIME - 28 DAYS, USA), WS-DATE-AREA, Y)
... WHERE RATE > 7

```

FD

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Purpose Define the file descriptions for input and output files, including report files.

Syntax: Format 1:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
FD      inputfilename|outputfilename
        [Applicable COBOL FD clauses]
```

Format 2:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
FD      filename
        [Applicable COBOL FD clauses]
        .
        .
        .
        REPORT IS|REPORTS ARE reportname1 [... reportname15]
```

- Comments:**
- Use one FD keyword per file description.
 - In Format 1, follow each file description with the file record description, using the *01*, *DS*, *REC*, or *++* keywords.
 - In Format 2:
 - Reportnames may be in any order.
 - Each Reportname must have a corresponding RED statement. Reportnames in both statements must be identical.
 - The RED statement for each report replaces the file record description entry required for Format 1.
 - The default size for a report record is 133. To define a different report record size in your FD statement, calculate the size as follows, and code it in a RECORD CONTAINS clause.

Record Size = Report mock-up size (maximum 247 characters)
 + 1 byte for carriage control
 + 2 bytes for the CODE clause, if used.

- If you define the report record in the WRITE ROUTINE clause, the default size is either 248 or 250. See *WRITE ROUTINE*.
- APS generates the file description 01-level identifier(s) for each report output file(s); you need not code them.
- If reportname exceeds 20 characters, Report Writer creates an abbreviated record name, as follows--the first character of each hyphenated word in the data name (except the last word), a hyphen, the last word, a hyphen, and RECORD. For example,

Input REALLY-LONG-REPORT-NAME
Output RLR-NAME-RECORD

Example: Report example:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
FD      INPUT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 0 RECORDS.
01      PART-STOCK-REC          PIC X(80).
FD      REPORT-OUTPUT-FILE
        LABEL RECORDS ARE STANDARD
        REPORT IS STOCK-REPORT.
```

Field Edits

Category: Screen Painter feature

Description: Screen fields can be one of the following types:

- Character
- Numeric
- Date or time

Define the internal, input, and output data representation

Depending on the type of field you create, you assign field edits that define the data representations for the field. APS supports the following field edits.

- Internal field edit values specify the COBOL picture characteristics for storing data the end user enters. These include:
 - Alphabetic, character, or numeric data type
 - Internal length
 - Binary or packed format
 - Sign specification
 - Right justification
- Input field edit values specify format and data requirements that the end user must adhere to when entering data into the field. These include:
 - Required data
 - Input mask
 - Julian, Gregorian, system, or user-defined date type
 - User-defined or system time type
 - Minimum/maximum input requirements
 - Testing for blank spaces or special characters
 - Testing for numeric data
 - Testing for specific values
 - Zero-fill when blank
- Output field edit values specify the format requirements for displaying the data. These include:
 - Output mask
 - Output picture
 - Julian, Gregorian, system, or user-defined date type
 - User-defined or system time type
 - Right justification
 - Commas
 - Zero suppression

- Floating, leading, and trailing symbols
- Values or conversion values ensure only certain values are entered. You can test for:
 - A specific value or range of values for input data
 - Conversion values for input and output data representation to define how data is stored and converted for output
- Application edit routines specify additional edits or tests for input or output data. Edit routines can be paragraphs, subprograms, or APS macros that you create. Or, you can select a predefined application edit routine from a centralized listing of edit routines maintained by your APS Administrator.

Specify global or specific error messages

When users fail to enter data correctly, you can display an error message that explains the problem. You can define two error messages, one for when the end user enters invalid data, and one for when the end user neglects to enter data that is required.

These can be global messages that display for all screen fields, or specific messages that display for individual fields. Messages defined for a specific field override the global messages. You can also define conditions for bypassing input edits under certain conditions.

Related Topics:	See...	For other information about Field Edits...
	<i>Field Edit Values</i> <i>Date and Time Field Edits</i>	Assigning the internal picture, input format and data requirements, and output display format for: <ul style="list-style-type: none"> • Character and numeric fields • Date and time fields
	<i>Application Field Edit Routines</i>	Specifying editing and testing routines for input or output data
	<i>Error Processing Messages</i>	Displaying error messages for invalid field data
	<i>Values, Conversion Values, and Value Ranges</i>	Ensuring that fields accept only certain data values

Field Edit Values

Category: Screen Painter feature (see *Field Edits*)

Description: Specify the data storage requirements, the format and data requirements that the end user must adhere to when entering data into the field, and specify the format requirements for displaying the data.

Procedure: Follow these steps.

- 1 From the Screen Painter, access the Field Edit Facility.
- 2 From either the Field Selection or Edit Selection screen, access the Internal Picture, Input Editing, or Output Editing screen.
- 3 Assign data representations by completing the fields listed below.
- 4 From either the Input Editing or Output Editing screen, you can transfer to:

Values or
Conversions

Enter **s** to transfer to the Values or Conversions screen to specify valid values, ranges of values, or conversion values. See *Values, Conversion Values, and Value Ranges*.

Application Edits

Enter **s** to transfer to the Application Editing screen to specify your own edits in a paragraph, subprogram, or APS macro. See *Application Field Edit Routines*.

Error Processing

Enter **s** to transfer to the Error Processing screen to specify error messages and attributes that display if the field fails input edits. See *Error Processing Messages*.

Internal Picture

Option

Description

Data Type

- | | |
|---|---------------------------|
| A | Aphabetic field. |
| C | Default. Character field. |
| N | Numeric field. |

Option	Description
	G For KANJI use only. Extended Graphics Character Set (EGCS).
Internal Length	Enter the number of characters. The default is the screen field length. For numeric fields, enter the number of digits that precede the decimal point.
Justified Right	Type s to generate right justification on the COBOL picture.
Decimal Places	Enter the number of digits that follow the decimal point.
COMP (Binary)	Type s to store input data in binary format. Not valid with signed data.
COMP-3 (Packed)	Type s to store input data in packed format. Not valid with signed data.
Signed	Type s to store the input data with either a positive or negative value. Not valid with a binary or packed format.
Sign Leading	Type s to store the sign at the left of the number. Not valid with a binary or packed format.
Sign Separate	Type s to store the sign in a separate byte from the number. Not valid with a binary or packed format.

Input Editing

Field	Description
Internal Picture	Enter s to transfer to the Internal Picture screen to change the storage format.
Required	Enter s to indicate that the end user must enter a value in the field.
Input Mask	Enter the pattern or mask to accept input data and separators. Or, enter s to transfer to the Masking screen and specify the mask in the Input Mask field. Note the following. <ul style="list-style-type: none"> Strip special characters on input if the internal picture does not have space for them. To remove special characters when converting the data to the storage format, enter s in the Strip Special Characters field on the Masking screen. A special character

Field	Description
	<p>is any character other than input mask characters.</p> <ul style="list-style-type: none"> • Include optional mask characters on either end of the mask, not on both ends. • Required mask characters do not require data entry; they require that any data is entered be of that specific type. For example, the A mask character requires an alphabetic character in that position if any data is entered <p>Note the following for character fields.</p> <ul style="list-style-type: none"> • The mask length, the mask characters plus special characters, must equal the field length. • The number of mask characters, excluding special characters, must equal the internal picture length. <p>Note the following for numeric fields.</p> <ul style="list-style-type: none"> • Valid mask characters are 9 and N. • The mask length cannot exceed the internal picture length. If it is less, APS places leading zeroes in the internal picture. • Include optional mask characters on either end of the mask, not on both ends.
Minimum Input	Enter the length of the shortest valid entry. Default is zero. You cannot specify an input mask with this option. This option does not imply that data is required.
Maximum Input	Enter the length of the longest valid entry. Default is field length. You cannot specify an input mask with this option. This option does not imply that data is required.
No Embedded Spaces	Enter s to reject characters separated by spaces. Leading and trailing blank spaces are unaffected by this option.

Field	Description
Numeric Test	Enter s to allow only numeric data.
Numeric De-Edit	Enter s to validate that the data is numeric and to remove special characters. Select this option if you specify an output COBOL picture; doing so removes the special characters for data computations. This option does not imply that data is required. Not valid with an input mask.
Zero When Blank	Move zero to the internal picture if no data is entered. Not valid with required fields or fields with input masks. To prevent zeroes from appearing in the field when data is not entered, zero suppress the output picture on the Output Picture screen.
Minimum Digits	Enter the smallest number of digits required before the decimal point. Default is zero. You cannot specify an input mask with this option. This option does not imply that data is required.
Minimum Decimals	Enter the smallest number of digits required to follow the decimal point. Default is zero. This option does not imply that data is required.
Maximum Digits	Enter the largest number of digits allowed before the decimal point. Default is the maximum number that fits in the internal picture. Not valid with an input mask. This option does not imply that data is required.
Maximum Decimals	Enter the largest number of digits allowed to follow the decimal point. Default is the maximum number that fits in the internal picture. This option does not imply that data is required.
Output Editing	
Field	Description
Internal Picture	Enter s to transfer to the Internal Picture screen to change the storage format.

Field	Description
Output Mask	<p>Enter the pattern or mask to position data and separators. Or, enter s to transfer to the Masking screen and specify the mask in the Output Mask field. Note the following.</p> <ul style="list-style-type: none"> • The X character is the placeholder for output data. • The mask length must equal the screen field length; the total number of Xs must equal the internal picture length.
Output Picture	<p>Enter the output COBOL picture. Or, enter s to transfer to the Output Picture screen and specify the mask in the Picture field. You cannot assign an output picture if you use an output mask.</p>
Right Justify	<p>Enter s to generate right justification for the output format.</p>
Insert Comma(s)	<p>Enter s to format data with commas in appropriate positions.</p>
Zero Suppression	<p>Enter s to generate the zero suppression symbol.</p>
Floating Symbol	<p>Enter a \$, +, or - symbol to generate a floating dollar, plus sign, or minus sign to the left of the first digit.</p>
Fixed Leading Symbol	<p>Enter a \$, +, or - symbol to generate a fixed dollar, plus sign, or minus sign in the leftmost position.</p>
Fixed Trailing Symbol	<p>Enter a \$, +, or - symbol to generate a fixed dollar, plus sign, or minus sign in the rightmost position.</p>

- Comments:**
- When you change the internal picture of a field with existing edits, a message warns you of the possible effects of changing the internal picture for that field. Press Enter or F3 to proceed with the change anyway.

- APS verifies an edit mask depending on whether the field has optional characters in an edit mask, as follows.

Optional Mask Characters	Verification
No	Left to right scan
Yes, on the left	Left to right scan
Yes, on the right	Right to left scan

- To reverse commas and decimal points in the output picture of numeric fields, such as ZZ.ZZZ.ZZ9,99, modify the CNTL(APSPROJ) file in one of the following ways, and then compile the screen and program.
 - Code the following assignment statements in CNTL(APSPROJ), where *membername* is the USERMACS member name that contains the \$TP-SPECIAL-NAMES macro definition. APS includes the member for you. The \$TP-SPECIAL-NAMES macro generates the DECIMAL-POINT-IS-COMMA statement.

```
% &FE-DECIMAL-POINT-IS-COMMA = "YES"
% &FE-TP-SPECIAL-NAMES-MACMBR = "TPSPEC" | "membername"
```

Then, copy TPSPEC from &SSMAPS..CNTL to &SSMDSN..USERMACS. Modify the \$TP-SPECIAL-NAMES macro as desired to generate statements in the SPECIAL-NAMES paragraph.

- Code the following statement in CNTL(APSPROJ). The DC target epilogue macro generates the DECIMAL-POINT-IS-COMMA statement.

```
% &FE-DECIMAL-POINT-IS-COMMA = "YES"
```

Fields and Flags, Data Communication

CICS Invocation Mode

Description: To determine how to invoke a CICS program, APS/CICS provides the following 88-level flags to indicate the mode of program invocation.

```
TP-INVOCATION-MODE          PIC X(01).
88 TP-TRANSID-INVOKED      VALUE 'T'.
```

```

88 TP-PROGRAM-INVOKED      VALUE 'P' .
88 TP-SCREEN-INVOKED       VALUE 'S' .
88 TP-LINK-INVOKED         VALUE 'L' .

```

TP-TRANSID-INVOKED A transaction code entered on a blank screen invokes the program.

TP-PROGRAM-INVOKED An XCTL call from another program invokes the program.

TP-SCREEN-INVOKED The SEND call sends the screen and a PF key or ENTER key invokes the program.

TP-LINK-INVOKED A LINK call from another program invokes the program.

IMS DC Invocation Mode

Description: To determine how to invoke an IMS DC program, APS provides the following 88-level program support variable flags.

```

TP-INVOCATION-MODE      PIC X.
  88 TP-TRANSID-INVOKED  VALUE 'T' .
  88 TP-PROGRAM-INVOKED  VALUE 'P' .
  88 TP-SCREEN-INVOKED   VALUE 'S' .

```

TP-TRANSID-INVOKED A transaction code invokes the program; the input message contains no additional data.

TP-SCREEN-INVOKED Program reads an input message consisting of more than just the transaction code; often this is screen data.

TP-PROGRAM-INVOKED A program, rather than a terminal, sends the input message. Program-invoked mode applies only to conversational programs.

APS tests for a TRUE value to determine which invocation paragraph to perform.

ISPF Dialog Invocation Mode

Description: To determine how to invoke an ISPF Dialog program, APS provides the following 88-level flags.

```
TP-INVOCATION-MODE                PIC X(01).
88 TP-TRANSID-INVOKED              VALUE 'T'.
88 TP-PROGRAM-INVOKED              VALUE 'P'.
88 TP-SCREEN-INVOKED                VALUE 'S'.
88 TP-LINK-INVOKED                  VALUE 'L'.
```

TP-TRANSID-INVOKED	Provided for upward compatibility; has no function.
TP-PROGRAM-INVOKED	A LINK call from another program invokes the program.
TP-SCREEN-INVOKED	The SEND call sends the screen and a PF key or ENTER key invokes the program.
TP-LINK-INVOKED	Provided for upward compatibility; has no function.

VSAM Batch

Description: Use APS-defined data fields and S-COBOL flags in your program.

Name	Associated Calls	Description
APS-END-PROCESS	DB-PROCESS	Flag that terminates a DB-PROCESS loop. Example: TRUE RESET-OBTAIN
<i>ddname</i> -RRN	DB-OBTAIN DB-PROCESS DB-ERASE DB STORE	Field that controls relative record number (RRN) of a retrieved or stored RRDS file record; generates value for the RELATIVE KEY clause of the SELECT statement. Ddname is the external file name specified in the subschema. Generated field definition <i>ddname</i> -RRN PIC 9(09).

Name	Associated Calls	Description
RESET-OBTAIN	DB-OBTAIN DB-PROCESS	Flag that resets browse to beginning of file. Example: TRUE RESET-OBTAIN

RRDS and ESDS Support

In RRDS processing, APS/VSAM generates ddname-RRN, deriving the RRN (relative record number) of the current retrieved record. This allows access to a specific RRN. The field value is the RRN of each retrieved or stored record, and is updated after every successful READNEXT operation.

Assign a RRN value to ddname-RRN prior to a DB-STORE, if the SELECT statement ACCESS clause is either RANDOM or DYNAMIC.

An ESDS file is opened in I/O mode.

VSAM Online

Description: Use APS-defined data fields and S-COBOL flags in your program.

Name	Associated Calls	Description
APS-shortrecname-VAR	DB-OBTAIN DB-STORE DB-MODIFY DB-PROCESS DB-ERASE	Field that contains actual record length after a successful record retrieval (direct or sequential for DB-OBTAIN and DB-PROCESS). Supplies value to dataarea in CICS LENGTH option. Shortrecname comes from the subschema definition; it is the REC card SHORT keyword. See also "Skip Sequence Processing" below.
APS-END-PROCESS	DB-PROCESS	Flag that terminates DB-PROCESS loop. Example: TRUE RESET-OBTAIN

Name	Associated Calls	Description
APS-VSAM-NUMREC	DB-ERASE	Field that contains number of records deleted after key-qualified ERASE with partial key length specified.
<i>ddname</i> -APS- <i>KEYnumber</i>	DB-OBTAIN DB-PROCESS	Field that contains APS-generated key name for use in skip-sequential processing. <i>Ddname</i> is the subschema file. See also "Skip Sequence Processing" below. Example: ORDER-APS-KEY1
<i>ddname</i> -RBA	DB-OBTAIN DB-PROCESS DB-STORE	Field that contains relative byte address (RBA) of a retrieved, stored ESDS file record; supplies value to CICS RIDFLD option. See also "ESDS Support" and "Skip Sequence Processing" below. <i>Ddname</i> is the subschema file. Generated field definition <i>ddname</i> -RBA PIC S9(08) COMP
<i>ddname</i> -RRN	DB-OBTAIN DB-PROCESS DB-STORE	Field that contains relative record number (RRN) of a retrieved or stored RRDS file record; supplies value to CICS RIDFLD option. See also "RRDS Support" and "Skip Sequence Processing" below. Generated field definition <i>ddname</i> -RRN PIC S9(08) COMP
RESET-OBTAIN	DB-OBTAIN DB-PROCESS	Flag that resets browse to beginning of file. When used with PREV, resets to end of file. Example: TRUE RESET-OBTAIN

ESDS Support

Ddname-RBA provides access to a specific relative byte address (RBA). The current retrieved or written (stored) record determines the RBA; every successful READNEXT, READPREV, and WRITE operation updates the RBA. Note the following.

- DB-ERASE cannot delete a record from an ESDS file.
- The operator must be EQUAL or = for a direct or positional DB-OBTAIN and for a qualified DB-PROCESS.
- Assign *ddname*-RBA an RBA value prior to a DB-STORE or direct (key-specified) call.
- If a key qualification specifies a non-existent RBA, the ILLOGIC condition occurs (the DB status flag AB-ON-REC). The APS-supplied abnormal condition processing may need to be customized to access this condition.

RRDS Support

Ddname-RRN provides access to a specific relative record number (RRN). The current retrieved determines the RRN; every successful READNEXT and READPREV operation updates the RBA.

Assign *ddname*-RRN a RRN value prior to a DB-STORE or direct (key-specified) call.

Skip Sequence Processing

Skip-sequential processing moves new key search criteria to an APS-generated key while an active sequential DB-OBTAIN or DB-PROCESS browse is in progress. The following indicates the APS-generated keys by target.

File Type	Record Key
ESDS	<i>ddname</i> -RBA PIC S9(08) COMP.
RRDS	<i>ddname</i> -RRN PIC S9(08) COMP.
KSDS	<i>ddname</i> -APS-KEYnumber

Number indicates the key number, as follows:
 1 = Prime record key
 The remaining numbers correspond to the order of the DDI IDX subschema cards for that record.
 2 = First alternate index,
 3 = Second alternate index, . . .

Variable Length Files

APS/VSAM provides the APS-shortreclname-VAR field, which contains length information for variable length files, for each record definition in the subschema.

APS-shortreclname-VAR must be updated with the calculated record length prior to all DB-STORE and DB-MODIFY calls.

For example, the following updates APS-CUST-VAR with the actual length of CUST-RECORD prior to the DB-STORE.

```
APS-CUST-VAR = WS-FIXED-PORTION +
... (CUST-NBR-ORDERS * WS-OCCURS-LENGTH)
DB-STORE REC CUST-RECORD
```

Field/Screen Cross Reference (SC02)

Category: APS-generated report (see *Application Reports*)

Description: The Field/Screen Cross Reference Report lists all the screens containing I/O fields, along with the attribute values for the field in each screen. The fields appear alphabetically on the report. For each field, the report lists the field attributes assigned in the Screen Painter, and the number of field occurrences within a repeated block. The reports denotes default values with periods (.). The end of the report shows the total number of fields cross-referenced, the associated screens reported, and the actual number of stored screens. The report documents this aspect of your application to support future maintenance and enhancement efforts.

- Comments:**
- Produce the Field/Screen Cross Reference Report from the Documentation Facility.
 - To select all fields, leave all fields on this screen blank.
 - To select a specific field, type its name in the Equal field.
 - To select a range of fields, select the subset you need by entering a Greater value, a Less value, or both. Use alphabetic characters to indicate the range; full field names are not necessary. For example,

FRFM

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Description: Pass COBOL or S-COBOL statements to the APS Generator without translation, in a freeform manner, and insert the statements in the program section where they are coded.

Syntax: Format 1, for Working-Storage and Linkage Sections only:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
FRFM   COBOLstatements|S-COBOLstatements
```

Format 2, for Procedure Division only:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
ENTER COBOL|ENTER S-COBOL|++INCLUDE membername
COBOLstatements|S-COBOLstatements
```

- Comments:**
- The next keyword (except the comment keyword, /*) terminates the FRFM function.
 - In Format 1, all COBOL and S-COBOL statements shift four spaces to the left during generation.
 - In Format 2, omit the FRFM keyword; all COBOL statements remain in Column 12 during generation.
 - Do not use the ENTER COBOL and ENTER S-COBOL statements within REPEAT or IF/ELSE-IF/ELSE structures.

Examples: Code 77-level data structures and put the data elements in column 8:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
FRFM   77 FIELD-A           PIC S99 COMP-3.
        77 FIELD-B           PIC S99 COMP-3.
        77 FIELD-C           PIC S99 COMP-3.
```

Code 77-levels with Customizer symbols:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYWS   &08+77 FIELD-A       PIC S99 COMP-3.
        &08+77 FIELD-B       PIC S99 COMP-3.
        &08+77 FIELD-C       PIC S99 COMP-3.
```

Functions, SQL

Category: Database call keywords (see *Database Calls*)

Compatibility: SQL target

Description: Code SQL built-in and scalar functions as keywords for the database access calls DB-DECLARE, DB-OBTAIN, AND DB-PROCESS.

Syntax: For DB-DECLARE:

```
DB-DECLARE cursorname [correlname1.]copylibname-REC
... function1[()(expression)[,resultfield[,Y][]]]
... function2[()(expression)[,resultfield[,Y][]]]
      .
      .
      .
... functionN[()(expression)[,resultfield[,Y][]]]
... WHERE ...
```

For DB-OBTAIN:

```
DB-OBTAIN REC [correlname1.]copylibname-REC
... function1[()(expression)[,resultfield[,Y][]]]
... function2[()(expression)[,resultfield[,Y][]]]
      .
      .
      .
... functionN[()(expression)[,resultfield[,Y][]]]
... WHERE ...
```

For DB-PROCESS:

```
DB-PROCESS REC [correlname1.]copylibname-REC
... [DB-PROCESS-ID name]
... function1[()(expression)[,resultfield[,Y][]]]
... function2[()(expression)[,resultfield[,Y][]]]
      .
      .
      .
... functionN[()(expression)[,resultfield[,Y][]]]
... WHERE ...
```

- Comments:**
- If you code a column function, every column in the call must use column functions, unless you code *GROUP BY*.
 - You can mix scalar functions with individual column selections in a call.
 - If you apply a function against a single column, the result is returned to the host variable. To override the result, specify a result field in the call and either:
 - Create a field in Working-Storage with the same data type as its corresponding columns or expression result. For scalar functions, the data type is determined by the function used and its associated rules.
 - Enter Y after the result field in the column function to let APS create the field.
 - If you apply a function against multiple columns, a literal, or an expression for which there is no host-variable, declare a result field in Working-Storage. The result field name can be as large as 26 characters; use COBOL naming conventions.
 - If the function can return a null value, it requires an associated indicator variable as follows.
 - When APS creates the result field, it creates a null indicator variable with the name based in the result field, such as resultfield-IND.
 - If you create the indicator variable in Working-Storage, use the same naming convention so that APS can correctly name each indicator variable in the resulting SQL.
 - Limit resultfield to 26 characters, to allow room for -IND. Define resultfield-IND as PIC S9(4) COMP.
 - If you do not specify a result field with the COUNT function, the APS-created field APS-COUNT-ROWS returns the COUNT result.

Examples: Select the minimum unit base price and count the number of colors it finds for a given part number; create the result fields and appropriate indicator variables in Working-Storage.

```
DB-DECLARE D2MAST-CURSOR D2TAB-REC
... MIN( ( PM_UNIT_BASE_PRICE ) , WS-PM-UNIT-BASE-PRICE , Y )
... COUNT( ( DISTINCT PM_COLOR ) , WS-PM-COLOR , Y )
... WHERE PM_PART_NO = :WS-PART-NO
```

SQL code with MIN and MAX functions:

```
DB-OBTAIN REC D2TAB-REC
... MAX((PM_UNIT_BASE_PRICE), WS-MAX-PRICE, Y)
... MIN((PM_UNIT_BASE_PRICE * PM_UNITS), WS-MIN-RESULT, Y)
... WHERE PM_PART_SHORT_DESC = 'WIDGET'
```

Generated code:

```
EXEC SQL SELECT
    MAX(PM_UNIT_BASE_PRICE)
    MIN(PM_UNIT_BASE_PRICE * PM_UNITS)
    INTO :WS-MAX-PRICE, :WS-MAX-PRICE-IND
    :WS-MIN-RESULT :WS-MIN-RESULT-IND
    FROM AUTHID.D2MASTER
    WHERE PM_PART_SHORT_DESC = 'WIDGET'
END-EXEC.
```

SQL code with SUM and AVG functions:

```
DB-OBTAIN REC D2TAB-REC
... SUM(PM_UNITS)
... AVG((PM_UNITS), WS-AVG-UNITS, Y)
... WHERE PM_PART_NO = '23432'
```

Generated code:

```
EXEC SQL select
    SUM(PM_UNITS)
    AVG(PM_UNITS)
    INTO :D2TAB-REC.PM-UNITS :IND-D2TAB-REC.IND-PM-UNITS,
    :WS-AVG-UNITS :WS-AVG-UNITS-IND
    FROM AUTHID.D2MASTER
    WHERE PM_PART_NO = '23432'
END-EXEC.
```

SQL code with COUNT function:

```
DB-OBTAIN REC D2MASTER-REC
... MAX((PM-UNITS), WS-MAX-PM-UNITS)
... COUNT((*), WS-PM-COUNT-FLD)
... AVG((PM-UNIT-BASE-PRICE), WS-AVG-PRICE)
... WHERE PM-PART-SHORT-DESC='WIDGET'
... AND PM-COLOR='RED'
```

Generated code:

```
EXEC SQL select
    MAX(PM-UNITS)
    MIN(PM-UNIT-BASE-PRICE)
```

```

COUNT(*)
AVG(PM-UNIT-BASE-PRICE)
INTO WS-MAX-PM-UNITS WS-MAX-PM-UNITS-IND,
WS-PM-COUNT-FLD,
WS-AVG-PRICE WS-AVG-PRICE-IND
FROM AUTHID.D2MASTER
WHERE PM-PART-SHORT-DESC='WIDGET'
AND PM-COLOR='RED'
END-EXEC.

```

SQL code with DATE, TIME, and AVG scalar functions:

```

DB-OBTAIN REC D2INVEN-REC
... IN_PART_NO
... DATE(IN_DATE_LAST_UPDTE)
... TIME((IN_TIME_LAST_UPDTE),WS-TIME-RETURN,Y)
... CHAR((IN_DATE_LAST_ORDER,ISO),WS-CHAR-RETURN)
... IN_QTY_ONHAND
... WHERE IN_PART_NO = '23432'

```

Generated code:

```

EXEC SQL SELECT
  IN_PART_NO
  DATE(IN_DATE_LAST_UPDTE)
  TIME(IN_TIME_LAST_UPDTE)
  CHAR(IN_DATE_LAST_ORDER,ISO)
  IN_QTY_ONHAND
INTO :D2INVEN-REC.IN-PART-NO,
:D2INVEN-REC.IN-DATE-LAST-UPDTE
:IND-D2INVEN-REC.IN-DATE-LAST-UPDTE,
:WS-TIME-RETURN :WS-TIME-RETURN-IND,
:WS-CHAR-RETURN :WS-CHAR-RETURN-IND,
:D2INVEN-REC.IN-QTY-ONHAND
:IND-D2INVEN-REC.IN-QTY-ONHAND
FROM AUTHID.D2INVTRY
WHERE IN_PART_NO = '23432'
END-EXEC.

```

In this example, the following occurs.

- Field IN_DATE_LAST_UPDTE is of type DATE.
- Because the result field WS-TIME-RETURN is followed by Y and the TIME function can return a null value, APS creates both a result field with a picture appropriate for a TIME data type and a WS-TIME-RETURN-IND indicator in Working-Storage.

- Because the result field WS-CHAR-RETURN is not followed by Y and the char function can return a null value, both the field and WS-CHAR-RETURN-IND indicator variable were created in Working-Storage.

GENERATE

Category: Report Writer statement (see *Report Writer Structures* in your APS *User's Guide* chapter *Creating Reports with Report Writer*)

Compatibility: Batch environments

Description: Produce either a detail report or a summary report; test controls; generate control breaks; print totals, detail lines, headings, and footings; clear counters and accumulators; and, paginate the report.

Syntax: `GENERATE dataname|reportname`

Parameters: *dataname* Produce a detail report. Specify dataname in the TYPE clause as DETAIL.

reportname Produce a summary report that contains no detail lines. Use reportname only if the referenced report group description contains:

- A CONTROL clause
- At least one REPORT HEADING, REPORT FOOTING, CONTROL HEADING, CONTROL FOOTING, or DETAIL group
- No more than one DETAIL report group

Comments:

- On first GENERATE clause execution, APS saves the control values. During other GENERATE executions, APS tests these control values to determine control breaks until it detects one. When a control break occurs, APS saves the new set of control values.

- During report printing, APS processes PAGE HEADING and PAGE FOOTING report groups for each page.
- When the first GENERATE clause of a report executes, APS processes, in order, the report groups defined in the report description-- REPORT HEADING, PAGE HEADING, and CONTROL HEADINGS, (from major to minor order).
- When GENERATE dataname executes, APS processes the designated report group. When GENERATE reportname executes, APS performs certain steps to process a DETAIL report group.
- When a GENERATE clause other than the first one executes, APS locates control breaks. The rules for determining the equality of control data items are the same as for relation conditions. If a control break occurs:
 - CONTROL FOOTING, USE procedures and CONTROL FOOTING, SOURCE statements are able to access the control data item values.
 - APS processes the CONTROL FOOTING report groups in minor to major order. APS does not process the CONTROL FOOTING report groups of a higher level than where the control break occurs.
 - APS processes the CONTROL HEADING report groups in major to minor order. APS does not process the CONTROL HEADING report groups of a higher level than where the control break occurs.
- GENERATE processing can only occur after INITIATE processing and before TERMINATE processing, for the report.
- To generate large reports, enter bigrwt in the APS Parm field on the Generator Options screen.

Example: See the APS *User's Guide* chapter *Creating Reports with Report Writer*.

Generator Options

Category: Application generation

Purpose Define the development environment for application, program, and screen generation.

Procedure: To select generator options, follow these steps.

- 1 Access the Generator Options screen. To do so, from the APS Options Menu enter option 2 in the Option field. Alternatively, from any APS screen enter **opt 2** in the **Command** field. The Generator Options screen displays.
- 2 Set options appropriate for your environment as described below.

Option	Description and Values
Target OS	Operating system.
DC	Data communications target. For valid DB/DC combinations see <i>DB/DC Target Combinations</i> .
DB	Database target. For valid DB/DC combinations see <i>DB/DC Target Combinations</i> .
SQL	SQL target.
Job Class	Specify any job class valid at your site and known to the APS generators.
Msg Class	Site-specific.
Listgen	Yes Generate a listing of generated code. See the APS Error Messages manual for sample. No Default.
COBOL	Yes Save generated COBOL program source in the library or PDS appropriate for your DC target. For the complete list of libraries and PDSs. No Default
Object	Yes Save generated object code in appropriate library.

Option	Description and Values
	No Default.
MFS/BMS	Yes Save generated BMS or MFS mapsets in the GENBMS or GENMFS libraries.
	No Default.
GENSRC	Yes Save generated source code in the GENSRC PDS or data set.
	No Default.
User Help	Yes Enable generation of APS User Help Facility source files.
	No Default.
Job Dest	Site-specific.
CARDIN Member	Specify the CNTL library APSDBDC member.
Generate COBOL II	Yes Generate COBOL II source code.
	No Default.
COBOL Compiler	1 OS/VS COBOL (Generate COBOL II = No) 2 COBOL II 3 COBOL for MVS
CICS Release	Specify the CICS release at your site.
IMS Release	Specify the IMS release at your site.
SUPRA	Yes Pass SUPRA procedural statements through APS unchanged.
	No Processes SUPRA procedural statements.
APS Parm	Override the APS Parm field on the Precompiler Options screen. Display all options whose default values you have overridden in the Precompiler Options screen. You can temporarily override these values simply by overtyping them in this field, but changes made here are not saved; they remain in effect only until you exit APS.
COBOL Parm	Specify parameters or directives for COBOL compiler. See the COBOL Language Operating Guide for valid values.

- Comments:**
- To reinstate all options to their default installation values, enter reset in the Command field.

- If you enter yes in the COBOL-II field, APS includes the new COBOL/2 generator support as well as MVS COBOL/2 compiler support. If you want to continue using the COBOL/74 generator support in conjunction with the COBOL/2 compiler, a fix will be provided.

Example:

```

COMMAND ==> _
TARGET OS ==> OS2      {MVS, OS2, PCDOS, OS400, VSE}
      DC ==> CICS      {PM, IMS, CICS, DLG, DDS, MVS, or ISPF [prototyper]}
      DB ==> USAM      {IMS, DLI, USAM, IDMS, or OS4}
      SQL ==> XDB       {Blank, OS2DM, XDB, SQLDS, DB2, SQL400}

JOB CLASS ==>
MSG CLASS ==>
      JOB DEST ==>
      CARDIN MEMBER ==> APSDBDC
      GENERATE COBOL-II ==> NO      {Yes or No}
      COBOL-II COMPILER ==> YES     {Yes or No}
      CICS RELEASE ==>              {Blank, A or B}
      IMS RELEASE ==>              {Blank, A or B}
      SUPRA ==> NO                  {Yes or No}
      EBCDIC ==> NO                 {Yes or No}
      PC CICS ==> MFOCUS            {IBM, MFOCUS}
      CLIENT/SERVER ==> NO         {Yes or No}

LISTGEN ==> NO
COBOL ==> NO
OBJECT ==> NO
MFS/BMS ==> NO
GENSRC ==> YES
APS DEBUG ==> YES
USER HELP ==> NO

APS Parm ==> XLATE=SCD
COBOL Parm ==>
    
```

Generation Parameters, Screens

Category: Screen Painter feature

Description: Change parameter values that affect the screen in any environment, as desired. Applicable parameters and valid values on the Screen Generation Parameters screen are:

Parameter		Description and Values
Prt Asm Mac Expn	F	Default. Do not print expanded assembler macros.
	T	Print macros.
No Assembler END	F	Default. Do not generate an assembler END statement.
	T	Generate statement.
Retain Dataname	F	Default. Do not retain painted field names as assembler labels.

Parameter	Description and Values	
	T	Retain field names. Under BMS or MFS, duplicate or invalid names can occur due to the maximum number of characters that BMS and MFS allow.
Global Fld Unpro	F	Default. Do not unprotect all I/O fields for prototyping.
	T	Unprotect all I/O fields.
Exattr Modifbl	F	Default. Do not modify extended attributes at run time.
	T	Allow modification at run time; generate EXTATTR=YES and extra attribute bytes in DSECT.
		Anything specified in this field has no effect during prototyping.
Gen Panel KANA	F	For KANJI only. Default. Do not generate the KANA keyword in the (ISPF panel) body header statement to display Japanese Katakana.
	T	Display Japanese Katakana.
Unpro Fld Box	F	For KANJI only. Default. Do not enclose unprotected I/O fields with lines.
	T	Enclose unprotected I/O fields.
Kextattr Modifble	F	For KANJI only. Default. Do not modify Format and Ruledline attributes for text fields at run time.
	T	Modify at run time.
System Message	NO or blank	Default. Do not display system messages.
	YES or SYMSG	Display messages on last line of the screen, if space is available.
	<i>fieldname</i>	Display messages in <i>fieldname</i> .

Parameter	Description and Values
	YES, <i>row</i> , <i>length</i> YES, <i>row</i> YES,, <i>length</i> Display message of up to <i>length</i> characters on specified row. Row default is last line of screen. <i>Length</i> can be from 40 to 70 characters or up to 131 characters for MOD5 screens.
Intensity	Change the intensity of all text fields. N Default. Normal. B Bright.
Color	Change the color of all text fields. NU Neutral BL Blue PK Pink TQ Turquoise RD Red GN Green YL Yellow
Blink Rvideo Underline	Set only one field to T(rue) for text fields. Blinking, reverse video, and underline are mutually exclusive.

CICS Parameters

Parameter	Description and Values
Associated Trans	Specify an associated transaction ID; default is the first four characters of the screen. If more than one screen begins with the same four characters, you need to define a unique transid.
Mapset Name	Override an APS-generated mapset name; maximum seven characters. To generate a multiple-map mapset that includes some or all screens, assign the same mapset name to the applicable screens in the application

Parameter	Description and Values
	The default mapset name reflects the number of characters in the screen name, as follows. 4-character name: <i>screenname</i> SET 5-, 6-character name: <i>screenname</i> \$ 7-character name: <i>screenname</i> \$; the \$ replaces the seventh character
Line	Starting line of the map on the physical screen; default is 001; value cannot exceed the screen depth.

ISPF Prototype Parameter

Parameter	Description and Values
Associated Pgm	Name of the program receiving control from the screen; default program name is <i>screenname</i> .

IMS DC Parameters

Parameter	Description and Values
Device Type	Standard device characters for different model terminals and printers. Defaults are IBM-recommended device characters. See your IBM MFS or IMS installation manual for further details.
Cursor Feedback	<p>F Default. Do not define a field in the MID as the cursor feedback field.</p> <p>T Provide cursor information for input processing. To hold the information, APS appends two halfword binary fields to the screen record. <i>screen-CURSOR-ROW</i> and <i>screen-CURSOR-COL</i>.</p> <p>Cursor feedback fields do not affect output cursor positioning.</p>

Parameter	Description and Values
DIF-DOF Name	<p>Override APS-generated name. Default reflects the number of characters in the screen name, as follows.</p> <p>4-character name: <i>screennameDF</i> 5-, 6-character name: <i>screenname\$</i> 7-, 8-character name: <i>screenname</i> truncated to 6 characters</p>
Optional Fld Name	<p>Specify fieldname or MFS PFKEY to hold the trancode or operator logical paging command. Alternatively, enter *PF and assign the PF key value on the MFS Function Keys screen, or *TC and construct a trancode on the Trancode Construction screen.</p>
MID Segment Exit: Number Vector	<p>Generate the EXIT parameter on the MID segment statement with Number as the exit routine number and Vector as the exit vector number. Valid values are:</p> <p>Number: 0 to 127 Vector: 0 to 255</p>
Opr Logical Paging	<p>F Default. Do not request operator logical paging.</p> <p>T Request paging. Enter name of field that will contain the paging requests in the Optional Fld Name field.</p>
MID Name	<p>Override APS-generated name. Default reflects the number of characters in the screen name, as follows.</p> <p>4-character name: <i>screennameMI</i> 5-, 6-, 7-character name: <i>screennameI</i> 8-character name: <i>screennameI</i>; the I replaces the eighth character</p>
MID Default Values	<p>F Default. Do not treat initial values as default values for fields in the MFS-generated MID.</p> <p>T Treat initial values as default values.</p>

Parameter	Description and Values
MOD Name	Override APS-generated name. Default reflects the number of characters in the screen name, as follows. 4-character name: <i>screennameMO</i> 5-, 6-, 7-character name: <i>screennameO</i> 8-character name: <i>screennameO</i> ; the O replaces the eighth character
MOD Fill Char	Generate fill characters in the MOD segment statement. Valid characters are --, NULL, PT, C, or 'x', where x is any character value.
DSCA	Override the Default System Control Area default value of X'00A0'.
"Labeled" Screen	F Do not append screen name to the input message. T Append the screen name.
Lines Per Page	If device type is a printer, specify number of lines to print on a page.
Trancode: Literal	Specify any literal value as the trancode. Default is the screen name.

GROUP BY

Category: Database call clause (see *Database Calls*)

Compatibility: SQL target

Description: Apply column functions to data elements that are collected into groups and define a hierarchy for these groups.

- Comments:**
- Code GROUP BY with DB-DECLARE and DB-PROCESS only.
 - Each column name in the SELECT list must be in the GROUP BY statement, and vice versa.
 - Use the ORDER BY keyword with GROUP BY to sort rows.

- Separate each item in the GROUP BY and ORDER BY clauses with a space; commas are optional.
- Use the HAVING clause, which acts as a WHERE clause, with GROUP BY to identify or evaluate the groups you want to include. A HAVING clause
 - Directly follows the GROUP BY statement
 - Names a grouping column or column function with its search conditions or qualifications
 - Can link qualifications with the Boolean operators AND and OR
 - When coded with COUNT, can test the number of rows found for a group. COUNT(*) operator value

Examples: This statement:

```
DB-PROCESS REC EMPLOYEE-REC
... DB-PROCESS-ID D2EMP
... EMP-DEPT
... MAX(EMP-SALARY)
... WHERE EMP-NAME NOT NULL
... GROUP BY EMP-DEPT
... ORDER BY EMP-DEPT
```

Results in:

EMP-DEPT	EMP-SALARY
-----	-----
FIN001	66700.000
FIN201	45250.000
MKT001	72300.000
PAY001	68800.000
PAY002	43500.000
SYS001	75000.000

This statement, which first groups rows by *EMP-DEPT* number, then within department by *EMP-SEX*, and then calculates maximum rates and salaries for each group:

```
DB-PROCESS REC EMPLOYEE-REC
... DB-PROCESS-ID D2EMP
... EMP-DEPT
... EMP-SEX
... MAX(EMP-RATE)
... MAX(EMP-SALARY)
... WHERE EMP-NAME NOT NULL
```

```

... GROUP BY EMP-DEPT,EMP-SEX
... HAVING COUNT(*) > 2
... AND MIN(EMP-RATE) > 3
... ORDER BY EMP-DEPT,EMP-SEX

```

Results in:

EMP-DEPT	EMP-SEX	EMP-RATE	EMP-SALARY
-----	-----	-----	-----
FIN001	F	32.067	66700.000
FIN001	M	31.105	64700.000
FIN201	F	19.711	41000.000
FIN201	M	21.754	45250.000
PAY001	F	33.076	68800.000
PAY001	M	31.884	66320.000
SYS001	F	32.692	68000.000
SYS001	M	36.058	75000.000

GSAM Calls

Category: IMS database calls

Compatibility: IMS DB target

Description: Access GSAM databases.

Syntax: IM-CLSE *view*
 IM-OPEN *view*
 IM-OPEN-INP *view*
 IM-OPEN-OUT *view*
 IM-OPEN-OUTA *view*
 IM-OPEN-OUTM *view*
 IM-GN *PCBname* [*ssa1* [...*ssa15*]]
 IM-GU *PCBname* *ssa1* [...*ssa15*]
 IM-ISRT *PCBname* [*ssa1* [...*ssa15*]]

Parameters: *pcbname* Database view; can be up to 20 characters.
 Default is IO-PCB.

<i>ssa</i>	SSA (record search) arguments that trigger COBOL MOVEs to or from the generated I/O area IM-IO-AREA. Default is IM-IO-AREA with a length of 32,000 bytes; set IM-IO-AREA-LEN in your program or DDISYMB member to override the length. Specify only one SSA per hierarchical level along a path. These arguments generate 8-byte records in Working-Storage to hold the corresponding values.
<i>view</i>	Database view. Must correspond to a GSAM PCB.

Example: The following code:

```
$IM-OPEN ("PARDBINP")
$IM-OPEN-INP ("PARDBINP")
$IM-GN ("PARDBINP")
```

Generates in the Procedure Division:

```
CALL 'CBLTDLI' USING
... IM-OPEN PARDBINP-PCB
MOVE PARDBINP-STATUS TO IM-STATUS
MOVE 'INP' TO IM-IO-AREA
CALL 'CBLTDLI' USING
... IM-OPEN PARDBINP-PCB
... IM-IO-AREA
MOVE PARDBINP-STATUS TO IM-STATUS
CALL 'CBLTDLI' USING
... IM-GN PARDBINP-PCB
... GSAM-IN-IO-AREA
MOVE PARDBINP-STATUS TO IM-STATUS
```

ID Parameters:

Category: Program Painter and Specification Editor keyword (see *Keywords:*)

Compatibility: IDMS DB target

Description: Code native IDMS calls in the Data and Environment Divisions and pass them through, without translation, to the precompile process.

Syntax: Data Division:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
IDCS   IDMSControlSectionstatements
      .
      .
      .
```

Environment Division:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
IDSS   SchemaSectionstatements
      .
      .
```

- Comments:**
- The IDCS keyword generates an IDMS-Control Section at the top of the Environment Division. All statements adjacent to or following this keyword and preceding the next keyword pass through to the generated program, starting in column 8.
 - The IDSS keyword generates a Schema Section at the top of the Data Division. All statements adjacent to or following this keyword and preceding the next keyword pass through to the generated program, starting in column 8.

IDM-COMMIT

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB target

Description: Write a COMT checkpoint to the IDMS journal file to designate the start or end of specific database accessing activities associated with the issuing run unit and release all record locks except select locks held on current records.

Syntax: IDM-COMMIT [ALL]

Parameter: ALL Release all record locks and set all currencies to null.

Example: Write a COMT checkpoint to the IDMS journal file; release all record locks held on current records; set all currencies to null.

```
IDM-COMMIT ALL
```

IDM-CONNECT

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB target

Description: Connect the named record to the named set.

Syntax: `IDM-CONNECT REC [recordname] TO setname`

Parameters: TO *setname*

Connect record to specified IDMS set name.

REC [*recordname*] Retrieve specified record. Define *recordname* as a member of the set and current of its record type. Put any area(s) that the set uses in an update mode.

- Comments:**
- Make the owner record current of its record type, and if needed, establish position by walking the set.
 - *Recordname* is optional because the record is previously located by a DB-OBTAIN REF.

Example Connect record type ORDER to CUSTOMER-ORDER. Note that owner record is current of record type.

```
MOVE SCR-ORDER-NUM TO ORDERNUM
MOVE SCR-QTY-ORD TO ORDER-QTY
DB-STORE REC ORDER
IF OK-ON-REC
    MOVE SCR-ORDER-NUM TO ORDER-NUM
    MOVE SCR-QTY-ORD TO ORDER-QTY
    DB-OBTAIN REC CUSTOMER WHERE CUST-NUMBER = WS-CUST-KEY
    IF OK-ON-REC
        IDM-CONNECT REC ORDER TO CUSTOMER-ORDER
```

```
IF AB-ON-REC
  PERFORM 950-ERROR-RTN
```

IDM-DISCONNECT

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB target

Description: Disconnect the record from the set in which it participates as an optional member.

Syntax: IDM-DISCONNECT REC [*recordname*] FROM *setname*

Parameters:

FROM <i>setname</i>	Disconnect record from specified IDMS set name.
REC [<i>recordname</i>]	Retrieve specified record. Define <i>recordname</i> as a member of the set and current of its record type. Put any area(s) that the set uses in an update mode.

Comment: *Recordname* is optional because the record is previously located by a DB-OBTAIN REF.

Example: Disconnect record type ORDER from CUSTOMER-ORDER.

```
DB-OBTAIN REF CUSTOMER REC ORDER
IF OK-ON-REC
  IDM-DISCONNECT REC ORDER FROM CUSTOMER-ORDER
IF AB-ON-REC
  PERFORM 950-ERROR-RTN
```

IDM-IF

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB target

Description: Determine if a data set is empty or if a record is a member of a given set; perform logic if condition is met.

Syntax: `IDM-IF SET setname [EMPTY|NOT EMPTY]
... [MEMBER|NOT MEMBER] paragraphname`

Parameters:

<code>[NOT] EMPTY</code>	Specify if there are member records for <i>setname</i> .
<code>[NOT] MEMBER</code>	Specify if the record, that is current of RUN-UNIT, is a member of <i>setname</i> .
<i>paragraphname</i>	Perform <i>paragraphname</i> when IF condition is met.
<code>SET <i>setname</i></code>	Specify IDMS set name.

Examples:

```
IDM-IF SET DEPT-EMPLOYEE MEMBER 2000-GET-OWNER

IDM-IF SET DEPT-EMPLOYEE NOT MEMBER
... 2100-NO-OWNER

IDM-IF SET DEPT-EMPLOYEE NOT EMPTY
... 3100-GET-EMPS
```

IDM-PROTOCOL

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB target

Description: Specify the program execution mode and the location of IDMS record descriptions.

Syntax: -KYWD- 12--*--20---*-----30-----*---40---*-----50---*-----60
 SYEN IDM-PROTOCOL *programmode location*

Parameters: *programmode* Execution mode. Refer to the IDMS COBOL Programmer's Guide for valid program modes.

location Place IDMS record descriptions copied from the IDMS Dictionary in this program area. Valid values are:

L-S or LS	Linkage Section
W-S or WS	Working-Storage
MANUAL	Generate an IDMS-Records Manual statement within Working-Storage; generate COPY and BIND statements for records referenced within the program.

- Comments:**
- APS generates IDM-PROTOCOL based on values found in the Application View, unless you code this call or code the IDCS keyword. The location for generated protocols is Working-Storage.
 - Code MANUAL to control DB-BIND processing.
 - If &IDM-GEN-AUTOSTATUS is set to YES in .CNTL(APIDMSIN), APS generates the appropriate autostatus protocol and copies the IDMS-STATUS paragraph to the program. For each DML command executed, APS checks the error status and performs IDMS-STATUS for an abnormal condition. Valid error status codes for DML commands are in .CNTL(APSIDMSIN).
 - If &IDM-GEN-AUTOSTATUS is set to NO, APS generates the appropriate non-autostatus protocol; the value of &IDM-GEN-IDMS-STATUS determines if IDMS-STATUS is copied to the program. No automatic error checking occurs.

Example: Set the protocol for an IDMS CICS program; place IDMS record descriptions in Working-Storage.

```
IDM-PROTOCOL CICS-EXEC-AUTO W-S
```

IDM-RETURN

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB target

Description: Return the database key and symbolic key (optional) from an indexed set via a currency or key value.

Syntax: `IDM-RETURN dataname FROM indexsetname
 ... [CURRENCY [FIRST|LAST|NEXT|PRIOR]]
 ... [USING keyfield]
 ... [KEY INTO keyfield]`

Parameters:	CURRENCY	Retrieve database key based on currency within the indexed set. If CURRENCY or USING is not coded, default is CURRENCY.
	<i>dataname</i>	Return the database key into the named Working-Storage field. Define as PIC S9(08) COMP SYNC.
	KEY INTO <i>keyfield</i>	Return the symbolic key obtained from the indexed set into Working-Storage keyfield. Currency specification is optional.
	USING <i>keyfield</i>	Use the value of <i>keyfield</i> to search the index and return the database key. See also "Comments:" below. If USING is not coded, default is CURRENCY.

Examples

```
IDM-RETURN WS-SAVE-DBKEY FROM IX-EMP-NAME
... CURRENCY

IDM-RETURN WS-SAVE-DBKEY FROM IX-EMP-NAME
... FIRST CURRENCY

IDM-RETURN WS-SAVE-DBKEY FROM IX-EMP-NAME

IDM-RETURN WS-SAVE-DBKEY FROM IX-EMP-NAME
... USING WS-EMP-KEY

IDM-RETURN WS-SAVE-DBKEY FROM IX-EMP-NAME
... FIRST CURRENCY KEY INTO WS-EMP-KEY
```

IDM-ROLLBACK

Category: Database call (see *Database Calls*)

Compatibility: IDMS DB target

Description: Write an ABRT checkpoint to the IDMS journal file to recover the recovery unit (the part of the run unit falling between two checkpoints); allow the run unit to continue accessing the database without issuing the DB-BIND and DB-OPEN calls.

Syntax: `IDM-ROLLBACK [CONTINUE]`

Parameter: CONTINUE Specify recovery unit is to roll back.

Comments:

- All currencies are nullified.
- If CONTINUE is coded, the recovery unit rolls back (recovers), but the run unit does not terminate. Database access resumes, without BIND and READY calls. Any character string is valid for a continue.

IDMS

Category: IDMS statement

Compatibility: IDMS DB target

Description: Code native IDMS calls in the Procedure Division and pass them through, without translation, to the precompile process.

Syntax:

```

-KYWD- 12-*----20---*----30---*----40---*----50---*----60
IDCS   IDMS nativecall
      .
      .

```

- Comments:**
- In the Procedure Division, begin every IDMS statement with the IDMS verb, typed in columns 12 to 72. Begin subsequent lines of the statement with a continuation ellipsis; do not use %
 - The IDMS Procedure Division statement generates, a \$IDMS rule. This rule generates IDMS COBOL at compile time. APS considers any code adjacent to the IDMS verb the first rule parameter, the next line the second parameter, and so on.
 - See also *ID Parameters*:

Example:

```
-KYWD- 12--*--20---*-----30-----*---40---*-----50---*-----60
        IDMS OBTAIN FIRST CUST-REC
        ... WITHIN CUST-REGION
```

Generated rule:

```
$IDMS ("OBTAIN FIRST CUST-REC",
% ... "WITHIN CUST-REGION")
```

IDMS DB Sample Programs

Compatibility: IDMS DB target

Batch Program:

```
-LINE- -KYWD- 12--*--20---*-----30-----*---40---*-----50---*-----60
000100  SYM1  % SET NOBLANK
000200          &07*NO-ACTIVITY-LOG
000300          &07*DMLIST
000800  PROC
001000          DB-OPEN MODE UPDATE
001100          DB-OBTAIN REC CUSTOMER FIRST AREA CUSTOMER-REGION
001300          IF OK-ON-REC                               /* APS IDMS 1.7 FLAG
001400          DISPLAY 'FIRST CUSTOMER IS ' CUST-NAME
001500          REPEAT
001600              DB-OBTAIN REC CUSTOMER
001601              ... NEXT AREA CUSTOMER-REGION
001700          UNTIL NOT OK-ON-REC
001701              DISPLAY 'NEXT CUSTOMER IS ' CUST-NAME
001710          IF END-ON-REC
001720              DISPLAY 'END OF AREA SWEEP'
001730          ELSE-IF VIO-ON-REC
```

```

001740             DISPLAY 'IDMS RULE VIOLATION ' ERROR-STATUS
001750         ELSE
001760             DISPLAY 'CATCH-ALL ERROR ' ERROR-STATUS
001770     ELSE-IF POS-ON-REC
001780         DISPLAY 'CANNOT POSITION WITHIN CUST AREA '
001781         ... ERROR-STATUS
001790     ELSE-IF VIO-ON-REC
001791         DISPLAY 'IDMS RULE VIOLATION OCCURRED '
001792         ... ERROR-STATUS
001793     ELSE
001794         DISPLAY 'CATCH-ALL ERROR ' ERROR-STATUS
001800     DB-CLOSE

```

- Line 100--The SYM1 keyword places FMP symbols and other statements before the Identification Division.
- Line 200--&07*NO-ACTIVITY-LOG is an IDMS translator directive to turn off the program statistics that are kept in the IDD. We recommend that you set the Dictionary Update parameter on the IDMS Options window.
- Line 300--&07*DMLIST is an IDMS translator directive, which sets the IDMS option to list the native DML calls prior to converting them. We recommend that you set the DMLIST parameter on the IDMS Options window.
- Line 800--The PROC keyword indicates that the following statements are procedural and generates the Procedure Division statement.
- Line 1000--DB-OPEN executes before any other database call. It translates into the IDMS/R READY command. This example readies the database in update mode.
- Line 1100--DB-OBTAIN retrieves the first customer record found within customer-region.
- Line 1300--IDMS flag OK-ON-REC tests for normal database conditions.
- Line 1600--DB-OBTAIN retrieves the next customer record found within the customer region.
- Line 1800--DB-CLOSE notifies IDMS that database processing for this program is complete, frees any locked records, and nullifies all database currency. If the program logic requires additional DB processing, an appropriate ready command must be executed. DB-CLOSE translates into the IDMS/R FINISH command.

Program with Generated Code:**Program Painter code:**

```

-KYWD- 12--*---20---*-----30-----*---40---*---50---*---60---*
IDCS
        PROTOCOL.      MODE IS BATCH DEBUG
                        IDMS-RECORDS IN WORKING-STORAGE SECTION
IDSS    DB SAMPSS WITHIN SAMPSSCH.
OPT     PROG
NTRY
PARA    MAIN-PARA.
        COPY IDMS SUBSCHEMA-BINDS
        IDMS READY USAGE-MODEL IS RETRIEVAL
        TP-PERFORM IDMS-STATUS
        TP-PERFORM PROCESS-CUSTOMERS
        IDMS FINISH
PARA    PROCESS-CUSTOMERS.
        IDMS OBTAIN FIRST CUST-REC WITHIN CUST-REGION
        IF ERROR-STATUS GREATER THAN '0000'
            TP-PERFORM ABORT-PARA
        ELSE
            TP-PERFORM DISPLAY-CUSTOMER
        REPEAT
            IDMS OBTAIN NEXT CUST-REC WITHIN CUST-REGION
        UNTIL ERROR-STATUS GREATER THAN '0000'
            TP-PERFORM DISPLAY-CUSTOMER
        IF ERROR-STATUS = '0307'
            DISPLAY '***** END OF CUST-REGION *****'
        ELSE
            PERFORM ABORT-PARA
PARA    DISPLAY-PARA.
        DISPLAY SPACE
        DISPLAY 'CUST-REC = ' CUST-REC
PARA    ABORT-PARA.
        DISPLAY SPACE
        DISPLAY '***** READ FAILURE *****'
        DISPLAY 'ERROR-STATUS = ' ERROR-STATUS
        IDMS FINISH
        STOP RUN

```

Generated S-COBOL program:

```

%      &AP-GEN-VER = 1719
%      &AP-PGM-ID = "TSTIDMS"
%      &AP-GEN-DB-TARGET = "VSAM"
%      &AP-TP-ENTRY-KYWD-SEEN = 1
%      &AP-SUBSCHEMA = ""
%      &AP-APPLICATION-ID = "TSTIDMS"

```

```

%   &AP-GEN-DATE = "870105"
%   &AP-GEN-TIME = "15482536"
IDENTIFICATION DIVISION.
PROGRAM-ID.                                TSTIDMS.
AUTHOR.                                    AP-SYSTEM GENERATED.
DATE-WRITTEN.                              870105.
DATE-COMPILED.                             &COMPILETIME.
ENVIRONMENT DIVISION.
IDMS-CONTROL SECTION.
PROTOCOL.    MODE IS BATCH DEBUG
              IDMS-RECORDS WITHIN WORKING-STORAGE SECTION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.                            &SYSTEM.
OBJECT-COMPUTER.                            &SYSTEM.
DATA DIVISION.
SCHEMA SECTION.
DB SAMPSS WITHIN SAMPSSCH.
WORKING-STORAGE SECTION.
$TP-WS-MARKER
$TP-COMMAREA
$TP-ENTRY (" ", " ")
MAIN-PARA.
    COPY IDMS SUBSCHEMA-BINDS
    $IDMS ("READY USAGE-MODEL IS RETRIEVAL")
    $TP-PERFORM ("IDMS-STATUS")
    $TP-PERFORM ("PROCESS-CUSTOMERS")
    $IDMS ("FINISH")
PROCESS-CUSTOMERS.
    $IDMS ("OBTAIN FIRST CUST-REC",
    % ... "WITHIN CUST-REGION")
    IF ERROR-STATUS GREATER THAN '0000'
        $TP-PERFORM ("ABORT-PARA")
    ELSE
        $TP-PERFORM ("DISPLAY-CUSTOMER")
    REPEAT
        $IDMS ("OBTAIN NEXT CUST-REC WITHIN CUST-REGION")
    UNTIL ERROR-STATUS GREATER THAN '0000'
        $TP-PERFORM ("DISPLAY-CUSTOMER")
    IF ERROR-STATUS = '0307'
        DISPLAY '***** END OF CUST-REGION *****'
    ELSE
        PERFORM ABORT-PARA
DISPLAY-PARA.
    DISPLAY SPACE
    DISPLAY 'CUST-REC = ' CUST-REC
ABORT-PARA.
    DISPLAY SPACE
    DISPLAY '***** READ FAILURE *****'

```

```

DISPLAY 'ERROR-STATUS = ' ERROR-STATUS
$IDMS ("FINISH")
STOP RUN

```

IDMS Options

Compatibility: IDMS DB target

Category: Application generation

Description: Define processing environment for application and program generation.

- Procedure:**
- 1 From the APS Options Menu enter option **7** in the **Option** field. Alternatively, from any APS screen enter **opt 7** in the **Command** or **Option** field. The IDMS Options screen displays.
 - 2 Specify IDMS options appropriate for your environment.

Option	Description and Values	
Dictionary Name	Specify the dictionary name.	
Central Version or Local	Specify the compile environment. APS generates a SYSTRNL with a unique DSN whose high level qualifier is your user ID.	
	cv	Default. Central Version.
	local	When you specify local, also enter a volume in the IDMS Local Jrnl Disk Vol field.
	dummy	When you specify <i>dummy</i> , APS generates a SYSTRNL DD DUMMY
IDMS Local Jrnl Disk Vol	Local compile disk volume for journal.	
Dictionary Update	Yes	Log program compile information to the dictionary.

Option	Description and Values	
	No	Default. Do not log program compile information.
IDMS DMLC Output to PDS	Yes	Write DMLC compile statements to a PDS. If you enter yes, you must allocate a &DSN..IDMSOUT PDS prior to compilation.
	No	Default. Do not write DMLC compile statements to a PDS.
IDMS Loadlib Qualifier	Specify full qualifiers for IDMS..LOADLIB.	
IDMS SYSCTL DSN	Optional. Specify DSN of IDMS dictionary.	
CV Node Name	Name of central version DDS (Distributed Database System) node under which loadlib program compiles.	
DMLIST (List Generation)	Yes	Generate list.
	No	Default.
Generate DB-BIND in Pgm	Yes	Generate the DB-BIND macro.
	No	Suppress the generation of the DB-BIND macro. Code the DB-BIND macro in your program.
IDMS Password	N/A	

IF/ELSE-IF/ELSE

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Evaluate a condition.

Reference

Syntax: Format 1:

```

IF condition1
  statementblock
[ ELSE-IF|ELSE IF condition2
  statementblock
  .
  .
  .
ELSE-IF|ELSE IF conditionN
  statementblock ]
[ ELSE
  statementblock ]

```

Format 2:

```

COBOLimperativestatement
... COBOLconditionalclause
  statementblock
ELSE-IF|ELSE IF condition1
  statementblock
[
  .
  .
  .
ELSE-IF|ELSE IF conditionN
  statementblock ]
[ ELSE
  statementblock ]

```

Logic Execution:

- When IF *condition1* is true, its statement block executes. Control then passes to the next statement at the same or less indented level as the IF statement (other than related ELSE-IF or ELSE statements which, by definition, are at the same level as their IF).
- When IF *condition1* is false, S-COBOL looks for ELSE-IF statements at the same level. If such ELSE-IF statements are present, the conditions are evaluated one after the other. If one of the conditions is true, its statement block executes. Control then passes to the next statement at the same or less indentation as the IF statement (other than related ELSE-IF or ELSE statements).
- If no ELSE-IF statements are present, or if the IF condition and the ELSE-IF conditions are all false, control passes to the statement block subordinate to the ELSE statement; if no ELSE statement is present, control passes to the next statement at the same or less indentation as the IF and ELSE-IF statements.

- If an ELSE-IF statement has no subordinate statement block and the condition is true, control passes to the next statement at the same or less indentation level as the ELSE-IF statement.

Comments:

- You can nest up to fourteen IF levels.
- The last IF, ELSE-IF, or ELSE coded in a related series requires a subordinate statement block.
- You can use the IF construct to code the NEXT SENTENCE concept of passing program control out of a particular construct to the next executable statement. See the example below.
- With Format 2, you can form conditional constructions similar to IF, ELSE-IF, ELSE by combining a COBOL imperative verb and its conditional clauses with ELSE-IF and ELSE statements. Verbs that permit this construction are

```
ADD, RETURN, CALL, REWRITE, COMPUTE, START, DELETE,
STRING, DIVIDE, SUBTRACT,
MULTIPLY, UNSTRING, READ, WRITE.
```

- AT END and INVALID KEY conditional clauses can be combined with ELSE-IF and ELSE statements.

Examples:

If line 2020 is false, pass control back to line 1020, the first statement after the PERFORM statement, because there is no ELSE-IF or ELSE coding associated with this IF, and the first character at the same or less indentation as this IF is a new paragraph name, which denotes the end of the preceding PERFORMed paragraph. If the line 2020 condition is true, execute its subordinate statement block (lines 2030 through 2170) and return control to line 1020.

After line 2030 is executed, test line 2040. If the condition in line 2040 is true, execute lines 2050 through 2071; pass control to line 2170. If line 2040 is false, test line 2080, etc. If lines 2040, 2080, and 2120 are false, execute the ELSE statement block (line 2160) and pass control to line 2170.

```
-LINE-  -KYWD-  12-*----20---*----30---*----40---*----50----
001010          PERFORM EMPLOYEE-BENEFIT-DEDUCTION
001020          MOVE ...
          .
          .
          .
002010  PARA    EMPLOYEE-BENEFIT-DEDUCTION
002020          IF EMPL-COVERAGE NOT = SPACES
```

```

002030          PERFORM CALC-BASIC-BEN
002040          IF EMPL-COVERAGE-TYPE = 'EXTRA'
002050              PERFORM CALC-EXTRA-BEN
002060              PERFORM CALC-DENTAL-BEN
002070              BEN-FIELD = XTRA-BEN +
002071                  ... DENTAL-BEN
002080          ELSE-IF EMPL-COVERAGE-TYPE =
002081              ... 'FAMILY'
002090              PERFORM CALC-FAMILY-BEN
002100              PERFORM CALC-DENTAL-BEN
002110              BEN-FIELD = FAMILY-BEN +
002111                  ... DENTAL-BEN
002120          ELSE-IF EMPL-COVERAGE-TYPE =
002121              ... 'DENTAL'
002130              PERFORM CALC-DENTAL-BEN
002140              BEN-FIELD = BASIC-BEN +
002141                  ... DENTAL-BEN
002150          ELSE
002160              BEN-FIELD = BASIC-BEN
002170          EMPL-DED-FIELD =
002171              ... BEN-FIELD * .5 / 12
002180
002190          PARA    CALC-BASIC-BEN

```

Nest conditional statements.

```

IF condition1
  statementblock1
  IF condition2
    IF condition3
      statementblock2
    ELSE-IF condition4
    ELSE-IF condition5
      statementblock3
    ELSE
      statementblock4
  ELSE
    statementblock5
    IF condition6
      IF condition7
        statementblock6

    statementblock7

```

Make the MULTIPLY function conditional by ON SIZE ERROR.

```

-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
          WS-NET-PAY = EMP-HOURS *
          ... EMP-HOURLY-RATE

```

```

... ON SIZE ERROR
    PERFORM PRINT-ERROR-MESSAGE
    DISPLAY SSNO WS-NET-PAY
    WS-NET-PAY WS-DEDUC = 0
ELSE-IF EMP-HOURLY-RATE =
... MIN-WAGE
    PERFORM CALC-DEDUC-MIN
ELSE-IF EMP-HOURLY-RATE < 5.00
    PERFORM CALC-DEDUC-1
ELSE-IF EMP-HOURLY-RATE >= 5.00
    PERFORM CALC-DEDUC-2
    IF EMP-HOURLY-RATE > 20.00
        DISPLAY SSNO
        ... EMP-HOURLY-RATE
ELSE
    DISPLAY SSNO EMP-HOURLY-RATE
    PERFORM PRINT-ERROR-MESSAGE
NET-PAY = WS-NET-PAY - WS-DEDUC

```

Using a NEXT SENTENCE construction, the following S-COBOL code:

```

-KYWD- 12-*----20---*----30---*----40---*----50---*----60
PARA   CALC-BENEFIT
        BEN-FIELD = ZERO
        IF PERM-PART-TIME
            PERFORM GROUP-A-CALC
            IF HRS-WORKED > 25
                BEN-FIELD =
                ... BEN-FIELD * 1.25
        ELSE-IF PART-TIME
        ELSE-IF FULL-TIME
            PERFORM GROUP-B-CALC
        EMPL-REC-BEN-FIELD =
        ... BEN-FIELD

```

Replaces the following COBOL code.

```

CALC-BENEFIT.
    MOVE ZERO TO BEN-FIELD.
    IF PERM-PART-TIME
        PERFORM GROUP-A-CALC
        IF HRS-WORKED > 25
            MULTIPLY BEN-FIELD BY 1.25
    ELSE
        NEXT SENTENCE
    ELSE
        IF PART-TIME
            NEXT SENTENCE
        ELSE

```

```

IF FULL-TIME
    PERFORM GROUP-B-CALC
ELSE
    NEXT SENTENCE.
MOVE BEN-FIELD TO EMPL-REC-BEN-FIELD.

```

\$IM- Data Communication Calls

Category: IMS Fast Path DC call (see *Data Communication Calls*)

Compatibility: IMS DC target for single-platform applications

Description: Perform Fast Path data communication calls.

Syntax:

```

$IM-CHNG altview [destination]
$IM-CMD [PCBname] [msgarea]
$IM-GCMD [PCBname] [msgarea]
$IM-GN PCBname SSA1 [... SSA15]
$IM-GU PCBname SSA1 [... SSA15]
$IM-ISRT [PCBname]altview SSA1 [... SSA15]
$IM-PURG [PCBname] [msgarea] [mod]

```

Parameters:	<i>altview</i>	Alternate view or IO PCB name.
	<i>destination</i>	Terminal destination; can be data name or literal in an 8-byte field.
	<i>msgarea</i>	Area where IMS returns the message segment being processed; APSAPS treats this area as the first segment of a new message.
	<i>mod</i>	Data name or literal in an 8-byte field naming the message output description.
	<i>pcbname</i>	Database view (maximum 20 characters); default is IO-PCB.

SSA Segment Search Argument, which triggers COBOL MOVEs to or from the generated I/O area IM-IO-AREA; default is IM-IO-AREA with a length of 32,000 bytes. To override the length, set IM-IO-AREA-LEN in your program or the DDISYMB member.

Specify only one SSA per hierarchical level along a path. These arguments generate 8-byte records in Working-Storage to hold the corresponding values.

Examples: The following code:

```
$IM-CHNG ("ALT-IO", "YOUR-TERMINAL-NAME")
```

Generates in Working-Storage:

```
01 IM-DESTINATION-NAME PIC X(8).
```

Generates in the Procedure Division:

```
MOVE YOUR-TERMINAL-NAME
... TO IM-DESTINATION-NAME
CALL 'CBLTDLI' USING
... IM-CHNG ALT-IO-PCB
... IM-DESTINATION-NAME
MOVE ALT-IO-STATUS
... TO IM-STATUS
```

The following code:

```
$IM-CHNG ("AOT-IO", "'YOURTERM'")
```

Generates in Working-Storage:

```
01 IM-DESTINATION-NAME PIC X(8).
```

Generates in the Procedure Division:

```
MOVE 'YOURTERM'
... TO IM-DESTINATION-NAME
CALL 'CBLTDLI' USING
... IM-CHNG ALT-IO-PCB
... IM-DESTINATION-NAME
MOVE ALT-IO-STATUS
... TO IM-STATUS
```

\$IM-FLD

Category: IMS Fast Path database call (see *Database Calls*)

Compatibility: IMS DB target

Description: Access Main Storage Data Bases (only) at the field level, and query the field contents and subsequently change the field value.

Syntax: `$IM-FLD MSDBview fsaname [rootssa]`

Parameters:	<i>fsaname</i>	COBOL dataname for the generated field search argument. See also <i>\$IM-FSA</i> .
	<i>MSDBview</i>	Main Storage Data Base PCB.
	<i>rootssa</i>	Root segment search argument; if not specified, APS retrieves the first segment in the MSDB.

Comment: Refer to your IMS documentation for more information on \$IM-FLD.

\$IM-FSA

Category: IMS Fast Path database call (see *Database Calls*)

Compatibility: IMS DB target

Description: Build your own field search argument (FSA) to use with the IM-FLD call.

Syntax: `$IM-FSA fsaname segment`
`... [field1[/picture/] operator operand1`
`... [field2[/picture/] operator operand2]`
`...`
`...`
`...`
`... [field10[/picture/] operator operand10]`

Parameters:	<i>field</i>	Segment field name; can be 8-character IMS field name or the corresponding COBOL record name.
	<i>fsaname</i>	COBOL data name for the generated FSA; references the FSA in \$IM-FLD.
	<i>operand</i>	Value to test against field in a field verify, or actual field value in a field change.
	<i>operator</i>	Valid operators for a field verify are: E Verify that <i>field</i> and <i>operand</i> are equal. G Verify that <i>field</i> is greater than <i>operand</i> . H Verify that <i>field</i> is greater than or equal to <i>operand</i> . L Verify that <i>field</i> is less than <i>operand</i> . M Verify that <i>field</i> is less than or equal to <i>operand</i> . N Verify that <i>field</i> is not equal to <i>operand</i> . Valid operators for a field change are: + Add <i>operand</i> from <i>field</i> . - Subtract <i>operand</i> from <i>field</i> . + Set <i>field</i> to value of <i>operand</i>
	<i>picture</i>	COBOL picture; if not specified, APS calculates picture from field length and field type.
	<i>segment</i>	The segment referenced by the FSA; can be eight-character IMS segment name or the corresponding COBOL field name.
Comment:	Refer to your IMS documentation for more information on \$IM-FSA.	

\$IM-POS

Category: IMS Fast Path database call (see *Database Calls*)

Compatibility: IMS DB target

Description: Access Data Entry Data Bases (DEDBs) only. Retrieve the location of a specific sequential dependent segment or the last inserted sequential dependent segment, or determine the amount of unused space within each DEDB area.

Syntax: \$IM-POS *DEDBview* [*SSA*]

Parameters: *DEDBview* Data Entry Data Base PCB
SSA Segment Search Argument, which triggers COBOL MOVEs to or from the generated I/O area IM-IO-AREA; default is IM-IO-AREA with a length of 32,000 bytes. To override the length, set IM-IO-AREA-LEN in your program or DDISYMB member.
 Specify only one SSA per hierarchical level along a path. These arguments generate 8-byte records in Working Storage to hold the corresponding values.

Comments: • After an IM-POS call successfully executes, information returns in the field IM-POS-IO-AREA, which is defined:

```

01  IM-POS-IO-AREA
    05  IM-POS-LENGTH                PIC S9(04) COMP.
    05  IM-POS-DATA-AREA-PONAME      OCCURS 240.
        10  IM-POS-AREA-DDNAME       PIC X(08).
        10  IM-POS-INFO
            15  IM-POS-CYCLS-COUNT    PIC S9(8) COMP.
            15  IM-POS-VSAM-RBA       PIC S9(8) COMP.
        10  IM-POS-UNUSED-SEQDEP-CIS PIC S9(8) COMP.
        10  IM-POS-UNUSED-INDEP-CIS  PIC S9(8) COMP.
  
```

• Refer to your IMS documentation for more information on \$IM-POS.

% INCLUDE

Category: APS Customizer statement (see your Customization Facility User's Guide)

Description: Open, read, and process a user-defined rule, copybook, or other file in an APS program.

Syntax: % INCLUDE *ddname(filename)*

Parameters: *ddname(filename)* Ddname and file to include

Comments:

- When including copybooks, use % INCLUDE when either of the following conditions are true.
 - You use a COBOL/2 compiler
 - Your copybook contains an indexed table
- When including copybooks, use a COBOL COPY statement when both of the following conditions are true.
 - You use an OS/VS COBOL compiler
 - Your copybook does not contain an indexed table
- In your program, you can code % INCLUDE in a file that is INCLUDEd; you can have up to ten levels of nested INCLUDEs.
- Always use an SY* keyword (see SY* Keywords)--such as SYM1--with an % INCLUDE to specify where to put the included file in the program.

Examples: Include a rule at the top of the program.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYM1   % INCLUDE USERMACS(MY-RULE)
```

Include a rule at the bottom of the program.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYBT   % INCLUDE USERMACS(MY-RULE)
```

Include a copybook in Working-Storage.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYWS   % INCLUDE COPYLIB(MY-COPYBOOK)
```

Include a copybook in Linkage.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYLK   % INCLUDE COPYLIB(MY-COPYBOOK)
```

INITIATE

Category: Report Writer statement (see *Report Writer Structures* and the *APS User's Guide* chapter *Creating Reports with Report Writer*.)

Compatibility: Batch environments

Description: Initialize all report counters and accumulators; set up control heading and control footing items.

Syntax: INITIATE *reportname1* [,*reportname2*] ...

- Comments:**
- Define each report name in a RED statement.
 - Initialization does not open the report file. Code a COBOL OPEN OUTPUT statement before the INITIATE to do so.
 - You must TERMINATE *reportname1* before executing an INITIATE for *reportname2*.

IO

Category: Program Painter and Specification Editor keyword (see *Keywords*.)

Compatibility: Batch environments

Description: Generate the Input-Output Section File-Control paragraph.

Syntax: -KYWD- 12-*----20---*----30---*----40---*----50---*----60
 IO *filename* ASSIGN [TO] *systemname*
 Applicable COBOL FILE-CONTROL clauses

- Comments:**
- Do not code the SELECT verb in your syntax.
 - Do not use ellipses to continue a FILE-CONTROL clause on subsequent lines. Any keyword except /* (the comment keyword) designates the end of the continuation.
 - Code an IO keyword for each file.
 - APS generates both the program INPUT-OUTPUT section header and the FILE-CONTROL header.

ISPF Dialog Compatibility: with IMS DC, CICS

Compatibility: ISPF Dialog target

Description The following calls improve communication between APS/ISPF Dialog and other DC targets.

NTRY	You can list explicit data areas; this allows flexibility in passing data areas between programs.
TP-COMMAREA	Not required to pass data areas between programs, but is supported for upward compatibility.
SEND	Provided for upward compatibility; does not invoke a program. Use LINK to invoke a program that displays a screen.
TP-LINKAGE	Provided for upward compatibility. This call is necessary for APS/CICS to order the LINKAGE section properly, but APS/ISPF Dialog lets you locate user records in LINKAGE without coding TP-LINKAGE.

The PFKEY option allows PF key processing similar to APS/CICS and APS/IMS with the Program-Controlled option when converting an APS/CICS or APS/IMS application to ISPF Dialog.

Screens being converted from APS/CICS or APS/IMS must have a command field, at least 4-bytes long, for users to enter ISPF commands and to allow PF key processing to function. The command field must be. Otherwise, data truncation errors result.

Do not use the APS Prototype Execution facility to execute your ISPF Dialog programs; execute them using the Dialog Test facility provided by ISPF.

Job Control Cards

Category: Application generation

Description: Create job cards to submit application/program generation batch jobs.

Procedure: To create up to five job cards--named J1 through J5-- with varying job names, account information, classes, and other attributes, follow these steps.

- 1 Access the Job Control Cards screen. To do so, from the APS Options Menu enter option **6** in the Option field. Alternatively, from any APS screen enter **opt 6** in the **Command** or **Option** field. The Job Control Cards screen displays.
- 2 Modify the cards as desired.

Joins

Category: Database access clauses

Compatibility: SQL target

Description: In the same call, select rows or specific columns from more than one table in the same call. Join tables together by using the DB-DECLARE, DB-OBTAIN, and DB-PROCESS calls.

Syntax: With DB-DECLARE:

```
DB-DECLARE cursorname correlname1.copylibname-REC
... [DISTINCT]
... [column1 [... columnN]]|[NONE]
      .
      .
      .
... correlnameN.copylibname-REC
... [column1 [... columnN]]|[NONE]
      .
      .
      .
... [WHERE correlname.column1 oper [:]value|correlname.column2
... [AND|OR correlname.column3 oper
[:]value|correlname.column4
      .
      .
      .
... AND|OR correlname.columnN operator [:]value|correlnameN]]
... [ORDER
... column1 [ASC|DESC] [...columnN [ASC]]]
```

With DB-OBTAIN:

```
DB-OBTAIN REC correlname1.copylibname-REC
... [DISTINCT]
... [column1 [... columnN]]|[NONE]
      .
      .
      .
... REC correlnameN.copylibname-REC
... [column1 [... columnN]]|[NONE]
... [WHERE correlname.column1 oper [:]value|correlname.column2
... [AND|OR correlname.column3 oper
[:]value|correlname.column4
      .
      .
      .
... AND|OR correlname.columnN oper [:]val|correlname.columnN]]
```

With DB-PROCESS:

```

DB-PROCESS REC correlname1.copylibname-REC
... [DB-PROCESS-ID name]
... [DISTINCT]
... [column1 [... columnN]]|[NONE]
      .
      .
      .
... REC correlnameN.copylibname-REC
... [column1 [... columnN]]|[NONE]
... [WHERE correlname.column1 oper [:]value|correlname.column2
... [AND|OR correlname.column3 oper
[:]value|correlname.column4
      .
      .
... AND|OR correlname.columnN oper [:]val|correlname.columnN]
... [DB-LOOP-MAX=number]
... [ORDER
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]
Controlled logic block

```

Parameters: See the applicable database call for keyword descriptions.

- Comments:**
- Including a correlation name with each column is optional when joining tables. Correlation names help insulate your code against changes in table structures, and emphasize the exact columns you are accessing. A call requires a correlation name if the column name in a select list or WHERE clause appears in more than one of the joined tables.
 - Use the NONE keyword to qualify on the table, that is, reference the table with a WHERE clause, but do not select columns.
 - Join your tables together before coding WHERE. You can join any number of tables. You can also join a table to itself.
 - The WHERE clause refers to any column prefaced with the correlation name assigned to its table. Be sure to enter the correct correlation name--the SQL Generator cannot check it for you.

Examples: Select specific columns from tables D2MASTER and D2INVTRY; add qualifications to the WHERE clause. Note that every column in the WHERE clause is preceded by a correlation name.

```

DB-OBTAIN REC A.D2TAB-REC
... PM_PART_NO PM_UNITS PM_COLOR

```

```

... REC B.D2INVEN-REC
... IN_PART_NO IN_QTY_ONHAND IN_DATE_LAST_ORDER
... WHERE A.PM_PART_NO = B.IN_PART_NO
... AND A.PM_COLOR = 'RED'
... AND B.IN_COLOR = 'RED'

```

Declare the cursor D2JOIN-CUR and include columns from tables D2MASTER and D2INVTRY. Use a WHERE clause to select rows where

- Columns PM_PART_NO and IN_PART_NO from each table match
- Column IN_QTY_ONHAND from table D2INVTRY is greater than 100

Eliminate duplicate rows from the cursor set (rows with matching data in every selected column are considered duplicates). Sort the cursor set in ascending order by PM_PART_NO from table D2MASTER, then in descending order by IN_QTY_ONHAND from table D2INVTRY.

```

DB-DECLARE D2JOIN-CUR A.D2TAB-REC DISTINCT
... PM_PART_NO PM_COLOR B.D2INVEN-REC
... IN_PART_NO IN_QTY_ONHAND
... WHERE A.PM_PART_NO = B.IN_PART_NO AND B.IN_QTY_ONHAND >
100
... ORDER A.PM_PART_NO B.IN_QTY_ONHAND DESC

```

Process the cursor D2MAST-ID. The cursor set includes columns from tables D2MASTER and D2INVTRY. Use a WHERE clause to include rows in the cursor set where:

- Columns PM_PART_NO and IN_PART_NO from each table match
- Column PM_PART_SHORT_DESC from table D2MASTER equals the Working-Storage variable :WS-PART-SHORT-DESC
- IN_QTY_ONHAND from table D2INVTRY is greater than 100

Eliminate duplicate rows from the cursor set (rows with matching data in columns PM_PART_NO, PM_COLOR, IN_PART_NO, and IN_COLOR are considered duplicates). Sort cursor set in ascending order by D2MASTER columns PM_PART_NO and PM_COLOR.

```

DB-PROCESS REC A.D2TAB-REC
... DB-PROCESS-ID D2MAST-ID
... DISTINCT
... PM_PART_NO PM_COLOR
... REC B.D2INVEN-REC
... IN_PART_NO IN_COLOR
... WHERE A.PM_PART_NO = B.IN_PART_NO
... AND A.PM_PART_SHORT_DESC =

```

```

... :WS-PART-SHORT-DESC
... AND B.IN_QTY_ONHAND > 100
... DB-LOOP-MAX=999
... ORDER A.PM_PART_NO ASC
... A.PM_COLOR ASC

```

Join a table to itself. Retrieve records where the IN_QTY_ONHAND column is greater than 100 and retrieve records whose IN_DATA_LAST_ORDER column matches these records.

```

DB-PROCESS REC A.D2INVEN-REC
... DB-PROCESS-ID D2MAST-ID
... IN_PART_NO
... IN_QTY_ONHAND
... IN_DATE_LAST_ORDER
... REC B.D2INVEN-REC
... IN_PART_NO      (:WS-PART-NO)
... IN_QTY_ONHAND   (:WS-QTY-ONHAND)
... WHERE A.IN_DATA_LAST_ORDER = B.IN_DATE_LAST_ORDER
... AND A.IN_QTY_ONHAND > 100

```

Keywords

Description: Program Painter/Specification Editor keywords designate program Divisions and Sections. The keyword you choose determines the location of your program code within the generated program.

Program Painter keywords also designate blocks of code within which all other categories of APS structures are coded. Following any keyword, you can enter

- Program code, including S-COBOL, COBOL, and COBOL/2 structures; database and data communication calls
- Data structures
- Report Writer structures
- User-defined rules

Code keywords in the Program Painter for batch, report, or complex online programs, or in Online Express.

Keywords: The following lists the keywords and their logical placement in a program.

Location	Keyword	Definition
Beginning of program	<i>SYM1</i> , see <i>SY* Keywords</i>	Define rule variables.
	<i>SYM2</i> , see <i>SY* Keywords</i>	Call user-defined rules after any rule libraries named in Application View.
IDENTIFICATION DIVISION	<i>REM</i>	Create COBOL comments in the Comments: Section. Not valid for COBOL/2.
ENVIRONMENT DIVISION	<i>SPNM</i>	Create Special-Names paragraph
	<i>SYEN</i> , see <i>SY* Keywords</i>	Call user-defined rules in Special-Names paragraph.
Input/Output Section	<i>SYIO</i> , see <i>SY* Keywords</i>	Call user-defined rules after any rule libraries.
	<i>IO</i>	Code SELECT statement.
DATA DIVISION	<i>SYDD</i> , see <i>SY* Keywords</i>	Call user-defined rules at the beginning of the Data Division.
File Section	<i>SYFD</i> , see <i>SY* Keywords</i>	Call user-defined rules after any rule libraries included.
	<i>FD</i> or <i>SD</i>	Code File Description or Sort Description statement.
Record description	<i>O1</i>	Code record description in COBOL format.
	<i>DS</i>	Specify name of data structure defined in Data Structure Painter containing record description.
	<i>REC</i>	Specify name of data structure defined in Data Structure Painter containing record description. Structure must be coded in Data Structure Painter format (see <i>Data Structures</i>).
	<i>++</i> , see <i>Joins</i>	Specify PANVALET member name containing record description.
WORKING-STORAGE	<i>SYWS</i> , see <i>SY* Keywords</i>	Call user-defined rules after any rule libraries and data structures named in Application View.

Location	Keyword	Definition
Data definitions	<i>WS</i>	Designate Working-Storage section.
	<i>01</i>	Code data structure in COBOL format.
	<i>DS</i>	Specify name of data structure defined in Data Structure Painter containing data definition.
	<i>REC</i>	Specify name of data structure defined in Data Structure Painter containing data definition. Structure must be coded in Data Structure Painter format (see <i>Data Structures</i>).
	<i>++ Joins</i>	Specify PANVALET member name containing data definition.
	<i>SQL</i>	Designate DB2 table or cursor.
LINKAGE SECTION	<i>CA</i>	Redefine the TP-USERAREA of the COMMAREA in CICS, IMS, and DLG parent programs.
	<i>SYLT, see SY* Keywords</i>	Call user-defined rules after any rule libraries and data structures named in the Application View and preceding any SYLK code.
	<i>SYLK, see SY* Keywords</i>	Call user-defined rules after any rule libraries and data structures named in the Application View.
Data definitions	<i>LK</i>	Designate Linkage Section.
	<i>01</i>	Code data structure in COBOL format.
	<i>DS</i>	Specify name of data structure defined in Data Structure Painter containing data definition.
	<i>REC</i>	Specify name of data structure defined in Data Structure Painter containing data definition. Structure must be coded in Data Structure Painter format (see <i>Data Structures</i>).

Location	Keyword	Definition
	<i>++</i> , see <i>Joins</i>	Specify PANVALET member name containing data definition.
	<i>SQL</i>	Designate DB2 table or cursor.
	<i>CA</i>	Redefine the TP-USERAREA of the COMMAREA in CICS, IMS, and DLG parent programs.
Report Section	<i>SYRP</i> , see <i>SY* Keywords</i>	Call user-defined rules after any rule libraries.
	<i>RED</i>	Create a Report Section and specify the report name.
	<i>MOCK</i>	Identify the report mock-up.
	<i>01</i>	Specify the type of report line, such as header, footer, detail.
PROCEDURE DIVISION	<i>NTRY</i> or <i>PROC</i>	Designate the Procedure Division; generate program skeleton and program invocation logic.
	<i>OPT</i>	Suppress generation of program invocation logic.
	<i>PARA and Paragraphs</i>	Designate S-COBOL or COBOL paragraph.
	<i>STUB</i>	Name the global code to be inserted.
Declaratives Section	<i>DECL</i>	Create a Declarative Section without paragraphs; code declarative statements.
	<i>DPAR</i>	Create a Declarative Section paragraph; code declarative paragraph statements.
Bottom of program	<i>SYBT</i> , see <i>SY* Keywords</i>	Call user-defined rules.
Anywhere in the program	<i>/*</i> , see <i>Comments</i>	Code comments. The <i>/*</i> keyword does not end the action of the current keyword; the next keyword ends the current one.
	<i>FRFM</i>	Pass associated COBOL or S-COBOL statements through the generator without translation and move the statements four columns to the left.

- Syntax: Rules**
- Code keywords in the Program Painter for batch, report, or complex online programs, or in Online Express.
 - Important A keyword is active until the appearance of another keyword, except for the comment keyword.
 - You can code keywords in any sequence. Remember, keywords designate blocks of logically related statement blocks and the action of one keyword is terminated by the appearance of the next keyword. The APS generators use these keywords to arrange the program according to COBOL program requirements.

Pass-Through Parameters:

The following keywords provide pass-through support by letting you code native calls and pass them through, without translation, to the precompile process.

IDCS and IDSS, see <i>IDMS</i>	Code and pass through native IDMS DB calls.
SQL	Code and pass through native SQL calls.
CICS	Code and pass through native CICS calls.

Example: The following examples include all APS keywords that you can enter in the Program Painter when creating online or batch programs. They illustrate

- The program locations where each keyword places the source code after you generate the program through APS to produce an executable COBOL or COBOL/2 program.
- The program locations at which APS places externally-defined components associated with your program, such as user-defined rules and data structures that you list on the Application View.

Online programs:

```
*SYM1 keyword places Customizer code here
*user rules from Appl View where Location=Top of program
*SYM2 keyword places Customizer code here
  IDENTIFICATION DIVISION.
      .
      .
      .
/* keyword places comments here (COBOL/2)
  REMARKS.  REM keyword places comments here (COBOL)
  ENVIRONMENT DIVISION.
      .
```

```

.
.
SPECIAL-NAMES.  SPNM keyword places code here
*SYEN keyword places Customizer code here
DATA DIVISION.
*SYDD keyword places Customizer code here
WORKING-STORAGE SECTION.
*user rules from Appl View where Location=Top of W-S
*data structures from from Appl View where Location=Top of W-S
*user rules from Appl View where Location=Working-Storage
*SYWS keyword places Customizer code here
*WS keyword, followed by any of the following six
*   keywords that you enter on the next line
*01 keyword
*REC keyword
*DS keyword
*SQL keyword
*++ keyword
*FRFM keyword
*user rules from Appl View where Location=Bottom of W-S
*CA, CA05, CADS keywords place code here in CICS, DDS, IMS,
*   DLG parent pgms
LINKAGE SECTION.
*user rules from Appl View where Location=Top of Linkage
*data structures from Appl View where Location=Linkage Section
*user rules from Appl View where Location=Linkage Section
*SYLT keyword places Customizer code here
*SYLK keyword places Customizer code here
*LK keyword, followed by any of the following six
*   keywords that you enter on the next line
*01 keyword
*DS keyword
*REC keyword
*SQL keyword
*++ keyword
*FRFM keyword
*CA, CA05, CADS keywords place code here in DLG child programs
*user rules from Appl View where Location=Bottom of Linkage
PROCEDURE DIVISION.  NTRY|PROC keyword generates this stmt
*OPT keyword places code here
.
.
.
*PARA keyword places code here
.
.
.
*STUB keyword places code here

```

```

.
.
.
*user rules from Appl View where Location=Bottom of program
*SYBT keyword places Customizer code here
*End of program.

```

Batch programs:

```

*SYM1 keyword places Customizer code here
*user rules from Appl View where Location=Top of program
*SYM2 keyword places Customizer code here
  IDENTIFICATION DIVISION.
.
.
.
/* keyword places comments here (COBOL/2)
  REMARKS.  REM keyword places comments here (COBOL)
  ENVIRONMENT DIVISION.
.
.
.
  SPECIAL-NAMES.  SPNM keyword places code here
*SYEN keyword places Customizer code here
  INPUT-OUTPUT SECTION.
*user rules from Appl View where Location=Top of I/O Section
*SYIO keyword places Customizer code here
  FILE-CONTROL.
*IO  keyword places code here
.
.
.
  DATA DIVISION.
*SYDD keyword places Customizer code here
  FILE SECTION.
*user rules from Appl View where Location=Top of File Section
*FD keyword places code here
*SD keyword places code here
  WORKING-STORAGE SECTION.
*user rules from Appl View where Location=Top of W-S
*data structures from from Appl View where Location=Top of W-S
*user rules from Appl View where Location=Working-Storage
*SYWS keyword places Customizer code here
*WS  keyword, followed by any of the following six
*    keywords that you enter on the next line
*01  keyword
*REC keyword

```

```

*DS keyword
*SQL keyword
*++ keyword
*FRFM keyword
*user rules from Appl View where Location=Bottom of W-S
LINKAGE SECTION.
*user rules from Appl View where Location=Top of Linkage
*data structures from Appl View where Location=Linkage Section
*user rules from Appl View where Location=Linkage Section
*SYLT keyword places Customizer code here
*SYLK keyword places Customizer code here
*LK keyword, followed by any of the following six
* keywords that you enter on the next line
*01 keyword
*DS keyword
*REC keyword
*SQL keyword
*++ keyword
*FRFM keyword
*user rules from Appl View where Location=Bottom of Linkage
PROCEDURE DIVISION. /*NTRY keyword generates this stmt
PROCEDURE DIVISION USING. /*PROC keyword generates this stmt
DECLARATIVES. /* DECL keyword generates this stmt
DECLARATIVES SECTION. /* DPAR generates this stmt
*DPAR keyword places code here
END DECLARATIVES.
*OPT keyword places code here
.
.
.
*PARA keyword places code here
.
.
.
*STUB keyword places code here
.
.
.
*user rules from Appl View where Location=Bottom of program
*SYBT keyword places Customizer code here
*End of program.

```

Limits

APS enforces the following size and programming limitations.

COBOL/2

Item	Max
Characters in paragraph name	24

Customization Facility

Item	Max
Indents	50
Nested macros	139
Macro call arguments	1000
Nested INCLUDEs	10

DECLARE statements

Subscripts	300
• Length of subscript	12
• Tables	200
• Parts per table	1000
• Length of a table part	78

System limits

• Work files (beginning with WORK4)	8
• LRECL for INCLUDE lib	80
• LRECL for extended INCLUDE lib	140

Online Express

Item	Max
Database calls	50
Record name occurrences	20
Field qualification occurrences	70

Item	Max
Scrolling for repeated blocks	1
Field mapping for repeated blocks	2

Subschema limits

• Files, databases, tables	130
• DB2 tables	99
• Records	160
• Qualifiable fields	990

Painters

Item	Max
Application Painter	
• Associated screens	30
• Associated data structures	60
• Associated USERMACs	90
Screen Painter	
• Number of fields	500
• ISPF only. Number 1-byte fields per screen	25
• ISPF only. Number of field attributes per screen	127
• Characters in field name	16
• Trancode construction fields	8
Secnario Painter: Fields per screen	400

Report Writer

Item	Max
Report mock-up lines	
• Coded in Copylib	200
• Coded in Report Painter	200
Number of reports for FD	15
Number SOURCE/SUM/ VALUE statements per program	300

S-COBOL

Item	Max
Paragraph	600
Characters in paragraph name	24
Indentation levels per nested IF structure	14
Paragraph arguments per program	400
EVALUATE statement	
• Conditional fields	255
• WHEN conditions	102
Symbol table entries (such as, paragraph names, file names, record names, verbs, section names, keywords, arguments, indexes, flags)	1801

LINK

Category: Data communication call (see *Data Communication Calls*)

Description: Transfer control and optionally send Commarea data from a CICS program at one logical level to a CICS APS or non-APS application subprogram; or, transfer control to an IMS or ISPF Dialog subprogram.

Syntax: CICS

Format 1, link to an APS program:

```
[TP-]LINK programname [errorpara]
... [DLIUIB pcbname [pcbname] ...]
... [userparm [userparm]...]
... [COMMAREA(dataarea) LENGTH(value)]|[NOCA]
```

Format 2, linking to a non-APS program:

```
[TP-]LINK programname(NONAPS)[errorpara]
... [COMMAREA(dataarea) LENGTH(value)]|[NOCA]
```

DDS

Format 1, CALL format:

```
[TP-]LINK programname [errorpara]
... [userparm [userparm]...]
... [COMMAREA]
```

IMS DC

```
[TP-]LINK subprogram [errorpara] [argument1 ... argument36]
```

DLG

Format 1, CALL format:

```
[TP-]LINK programname [errorpara]
... [userparm [userparm]...]
... [COMMAREA]
```

Format 2, SELECT format:

```
[TP-]LINK programname
... [errorpara]
... [options]
```

ISPF Prototyping

```
[TP-]LINK programname [errorpara]
... [COMMAREA(dataarea) LENGTH(value)]|[NOCA]
```

Parameters:	<i>argument</i>	Pass record area(s), such as a PCB. TP-COMMAREA, TP-USERAREA, or *NOSPA, may be an argument for conversational programs.
	COMMAREA	Pass the TP-COMMAREA, if the invoking program is the main program. Pass DLG-LINKAGE-COMMAREA, if the invoking program is a called program, that is, one where &DLG-COMMAREA-IN-LINKAGE = 'YES'. See also "Comments" below.
	COMMAREA (<i>dataarea</i>)	Pass data area to a program instead of TP-COMMAREA.
	DLUIB <i>pcbname</i>	DLI User Interface Block and the Program Control Block for the next program.

<i>errorpara</i>	User-defined error routine to perform when an abnormal condition occurs. <i>Errorpara</i> is positional; if omitted, code an asterisk (*) in its place.
LENGTH (<i>value</i>)	Maximum length of data; can be a literal or COBOL data name; define as S9(04)COMP.
(NONAPS)	Program is not an APS program.
NOCA	Do not pass COMMAREA.
<i>options</i>	ISPF Dialog Management services SELECT PGM options.
<i>programname</i>	Program name; can be a literal, variable, or combination, where the literal is moved to the variable (indicated by a slash (/) as the first character) by macro logic.
<i>subprogram</i>	APS-painted subprogram that links the calling program.
<i>userparm</i>	Pass linkage data area(s). Code with TP-LINKAGE, which names the 01-level user-defined area in the Linkage Section.

Comments: CICS

- When linking to an APS program, TP-COMMAREA passes by default. Alternately, you may pass another data area, or pass neither. Ensure that the Commarea in the "linked-to" program is the same length as the TP-COMMAREA or alternate data area you pass.
- When linking to a non-APS program, TP-USERAREA passes by default. Alternately, you pass another data area, or pass neither. Ensure that the Commarea in the non-APS program is the same length as the TP-USERAREA or alternate data area you pass. (TP-USERAREA gets its value from &TP-USER-LEN.)

Additionally, code the following before your NTRY statement:
 % &TP-PROGRAM-INVOCATION = "NONAPS"

- Coding TERM in the subprogram returns control to the program that issued the link call.

ISPF Dialog

- Including a TERM call in the invoked program returns control to the program issuing the LINK.

- Code *userparms* in the same order as those in the LINKAGE SECTION and the NTRY statement in the linked to program.
- To receive TP-COMMAREA as a passed data area from a called program, code the following before the NTRY keyword.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
      % &DLG-COMMAREA-IN-LINKAGE = 'YES'
```

- LINK generates a COBOL CALL or ISPEXEC SELECT, depending on the PROGRAM CONTROL TRANSFER option.
- The &APSPRE..CNTL member APDLGIN assigns the default value for the PROGRAM CONTROL TRANSFER option. The variable &DLG-PROGRAM-TRANSFER-OPTION can have the values CALL (default) or SELECT.
- The PROGRAM CONTROL TRANSFER option affects the availability of function pool variables between programs. Using CALL makes the current function variable pool available to the linked-to program; using SELECT does not.
- To override the PROGRAM CONTROL TRANSFER option:
 - a At the installation level, change the value in &APSPRE..CNTL(APDLGIN).
 - b At the project group level, code the assignment statement in &DSNPRE..CNTL(APSPROJ).
 - c At the application level, code the assignment statement in a &DSNPRE..USERMACS member and include it for all application programs.
 - d At the program level, code the assignment statement at the top of the program.
- The PROGRAM CONTROL TRANSFER option lets you invoke the LIBDEF service to define application-level libraries via the SELECT value.

Examples: CICS

Link to PGM001. No PSB is active and no arguments pass.

```
LINK PGM001
```

Link to the program stored in WS-PROGNAME. No PSB is active and no arguments pass.

```
LINK /WS-PROGNAME
```

Move the value PGM003 to WS-PROGNAME and link to that program.

```
LINK PGM003/WS-PROGNAME
```

Link to PGM004; execute an error routine for an abnormal condition. There is no active PSB and no arguments pass.

```
LINK PGM004 ERROR-PARA
```

Link to PGM005. Schedule a PSB and use a PCB named ABC-PCB for the linked-to program. Omit the positional parameter *errorpara* .

```
LINK PGM005 * DLIUIB ABC-PCB
```

Link to PGM006 and pass a Linkage data area SCR6-RECORD so that PGM006 can move data directly to screen fields.

```
LINK PGM006 * SCR6-RECORD
```

Link to PROG001, a non-APS program, passing TP-USERAREA only.

```
LINK PROG001(NONAPS)
```

Link to PROG002, a non-APS program, passing the data area WS-COMMAREA.

```
LINK PROG002(NONAPS) * COMMAREA(WS-COMMAREA) LENGTH(100)
```

Link to PROG003, a non-APS program, without passing a COMMAREA.

```
LINK PROG003(NONAPS) * NOCA
```

IMS DC

The following examples show the CALL statements generated in the program after linking to subprograms.

```
Coding      LINK SUBPGM * ARG1 ARG2 ARG3
Yields CALL 'SUBPGM' USING TP-COMMAREA ARG1 ARG2 ARG3
```

```
Coding LINK SUBPGM * *NOSPA ARG1 ARG2 ARG3
Yields CALL 'SUBPGM' USING ARG1 ARG2 ARG3
```

```
Coding LINK SUBPGM * ARG1 TP-USERAREA ARG2 ARG3
Yields CALL 'SUBPGM' USING ARG1 TP-USERAREA ARG2 ARG3
```

ISPF Dialog

Invoke PGM001 as a new program function. The SELECT service option NEWPOOL creates a new shared variable pool. The SELECT option is in effect.

```
LINK PGM001 * NEWPOOL
```

Invoke PGM001 as a new program function. The SELECT service option PARM passes data to PGM001 via LINKAGE SECTION. To pass variable data, VDEFINE a dialog variable. The SELECT option is in effect.

```
DLG-VDEFINE 01 WS-EMPL-NBR PIC X(05) AS EMPNBR
MOVE EMPL-NBR TO WS-EMPL-NBR
LINK PGM001 * PARM(' &EMPNBR ')
DLG-VDELETE EMPNBR
```

Link to the program in WS-PROGNAME. The CALL option is in effect.

```
MOVE 'PGM001' TO WS-PROGRAM
LINK /WS-PROGNAME
```

Link to PGM006 and pass Linkage data area SCR6-RECORD so that PGM006 can move data directly to screen fields. The CALL option is in effect.

```
LINK PGM006 * SCR6-RECORD
```

LK

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Description: Create or include data structures in the Linkage Section.

Syntax: -KYWD- 12-*----20---*----30---*----40---*----50---*----60
LK
kywd associated data structure

Parameter *kywd* Associated data structure keywords are: 01, DS, REC, See SQL, ++, or 01.

Example: Code Working-Storage and Linkage data structures using section keywords.

```

-KYWD- 12-*----20---*----30---*----40---*----50---*----60
IO      INPUT-FILE ASSIGN TO UT-S-INPUT
FD      INPUT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 0 RECORDS
01      INPUT-REC          PIC X(80) .
WS
REC     WS-INPUT-REC .
        WS-IN-PART-NO      N8
        WS-IN-DESC        X50
        WS-IN-BASE-PRICE  N6V2
++
LK
REC     LK-INPUT-REC
        LK-IN-PART-NO     N8
        LK-IN-DESC       X50
        LK-IN-BASE-PRICE N6V2
01      LK-OUT-REC .
DS05   LK-OUT-REC

```

Macro/Program Cross Reference (MC01)

Category: APS-generated report (see *Application Reports*)

Description: The Macro/Program Cross Reference Report lists the macros used in one or more applications, along with all of the programs that invoke the macros, and the macro libraries that the programs reside in. Use the report to find all the programs and applications that use a given macro.

The cross-referenced macros appear alphabetically within an application on the report. If you report on all applications, the applications display alphabetically in the report. Specific macro libraries are listed as follows.

- Macro libraries specified in the Application Painter display in the macrolib indicator column.
- Global macro libraries display *****ALL***** in the type indicator column.

- Stubs and batch programs display *****STUB***** in the type indicator column.

After each application, summary statistics provide the:

- Total number of macros and libraries reported
- Number of programs with macro calls
- Number of common macro libraries
- Total number of programs in that application

Comment: Produce the Macro/Program Cross Reference Report from the Documentation Facility.

Example:

```
REPORT CODE: MC01   APS APPLICATION PAINTER   PAGE 1
                MACRO/PROGRAM CROSS REFERENCE   01/17/92   08:26
                MKTAPS.MKT2
```

```
SELECTION CRITERIA: TDDEMO
APPLICATION: TDDEMO
```

```
*****
NAME OF MACRO / MACROLIB:   MACROLIB.   PROGRAM / STUB TYPE   NO.OF TIMES
                           IND.         USING MACRO IND       MACRO CALLED
-----
CLEAR                       TDCM                1
                           TDOM                1
                           TDPM                1

DB-ERASE                     TDCM                1
                           TDOM                1
    TDPM   1

.
.
.
PX-CA-COMPUTE-LEN           TDOM                1

.
.
.
TP-ATTR                     TDOM                13
                           TDOT                1
TP-PERFORM                   TDCM                36
```

Reference

	TDCS	27
	TDME	3
	TDOJ	27
	TDOM	80
	TDOT	16
	TDOU	9
	TDPF	7
	TDPL	27
	TDPM	36
.		
.		
.		
XCTL	TDCM	1
	TDCS	1
	TDME	7
	TDOJ	1
	TDOM	1
	TDOT	1
	TDOU	1
	TDPF	1
	TDPL	1
	TDPM	1
TOTAL MACROS & LIBRARIES REPORTED 14		
NO. OF PGMS. WITH MACRO CALL IN CODE 10		
NO. OF COMMON MACROLIBS IN APPLN. 0		
TOTAL NO. OF PROGRAMS IN APPLN. 10		

MFS Function Keys

Compatibility: IMS DC target

Category: Screen Painter feature

Description: Assign trancodes, operator logical paging, other IMS commands, and nulls or spaces to the MFS PF key fields.

Procedure: To define MFS function key values, follow these steps.

- 1 Type ***pf** in the **Optional Fld Name** field on the Screen Generation Parameters screen.

- 2 From the Screen Generation Parameters screen, enter **pf** or **pfk** in the **Command** field. The MFS Function Keys screen displays.
- 3 Enter ***null** or ***space** in the **PFkey Global Default** field to set a global PF key default.
- 4 Enter the PF key functions.
- 5 To check for errors in the function key values you entered, press Enter. A message displays for any value in error.
- 6 To save your key assignments and exit this screen, press F3. To exit without saving your key assignments, enter **cancel** in the **Command** field.

Example:

```

COMMAND ==>

                                     PFKEY GLOBAL DEFAULT:

PFK01: =*1                          PFK02: =-1
PFK03: /for frm1mo                   PFK04: /for frm12mo
PFK05: /for frm13mo                 PFK06: tran1
PFK07: tran2                         PFK08: tran3
PFK09: /exit                          PFK10: /end
PFK11:                                PFK12:
PFK13:                                PFK14:
PFK15:                                PFK16:
PFK17:                                PFK18:
PFK19:                                PFK20:
PFK21:                                PFK22:
PFK23:                                PFK24:

```

MFS Trancode Construction

Compatibility: IMS DC target

Category: Screen Painter feature

Description: Construct trancodes of up to eight parts by concatenating screen fields, literals, and PF keys.

Procedure: To create a trancode, follow these steps.

- 1 Enter ***tc** in the **Optional Fld Name** field on the Screen Generation Parameters screen.

- 2 From the Screen Generation Parameters screen, enter **tc** or **tran** in the **Command** field. The MFS Trancode Construction screen displays.
- 3 Complete the screen fields as follows.

Field	Description and Values
Field Name	Enter name of field or * pf . If you enter * pf , specify PF key values on the MFS Function Keys screen. If you specify a field, APS ignores all field attributes, including an initial value entered in the Value field. You must enter the initial value in the Default Literal field.
Default Literal	Enter any literal. If you supply a literal on the same line as a field name or * PFKEY , APS transmits the literal in the MID when no value is entered for the field or when the PF key is not invoked.
Fill	Enter the MID fill character for the associated field name or * PFKEY . S Default. Space N Null

- 4 To check for errors in the trancode values you entered, press Enter. A message displays for any value in error.
- 5 To save your trancode and exit this screen, press F3. To exit without saving your key assignments, enter **cancel** in the **Command** field.

Example: Define a trancode definition, consisting of a function key (values specified in the MFS Function Keys example on *Example:*), and the fields OPTION-PREFIX and NEXT-OPTION. If the end user enters no values in these two fields, the literal UPDTMENU transmits in the MID.

```

COMMAND ==>

          FIELD NAME          DEFAULT LITERAL    FILL
TRANCODE PART 1 - *pfkey          n
TRANCODE PART 2 - option-prefix  updt
TRANCODE PART 3 - next-option   menu_
TRANCODE PART 4 -
TRANCODE PART 5 -
TRANCODE PART 6 -
TRANCODE PART 7 -
TRANCODE PART 8 -

```

MID MOD Reorder

Compatibility: IMS DC target

Category: Screen Painter feature

Description: Change the MID/MOD order in which the I/O fields from your screen appear in the generated Working-Storage. Do so to meet any site standards that require transmittal of screen data in a particular order.

Procedure To change the order in which your I/O fields appear in the generated Working-Storage, follow these steps.

- 1 From the Screen Generation Parameters screen, enter **mm** or **mid-mod** in the **Command** field. The MID MOD Reorder screen displays, with I/O fields listed in the order they appear on the application screen, numbered by increments of ten.
- 2 Reorder fields as follows.
 - Type new sequence numbers, followed by at least one space, over the existing ones. A trancode field is always the first field, regardless of its sequence in the painted screen or on this screen.
 - Enter **reorder** in the **Command** field. Reordering fields in a repeated record block reorders the entire block.
- 3 To reset the sequence of fields to the sequence on your screen, enter **reset** in the **Command** field.
- 4 To save your new sequence and exit this screen, press F3. To exit without saving your sequence, enter **cancel** in the **Command** field. Any fields you add later to your screen are placed at the end of the modified MID MOD sequence list.

Example: Reorder the fields in the first screen. The second screen sequences ERR-MSG after PAGING in the Working-Storage Section. The third screen shows the result.

```

COMMAND ==>
ORDER  NAME      ROW  COL  LEN  TP  SEGMENT THRESHOLD -
00010 PART-NBR      06  019 008  U   IN  SCR-ORDER  SEG
00020 SHORT-DESC   08  019 031  P   N   001
00030 RB STARTS ROW 12  000 000  P   N   002
00040 ERR-MSG      20  002 074  P   B   003
00050 PAGING       21  002 008  P   D   004

```

```

COMMAND ==> reorder_
ORDER  NAME      ROW  COL  LEN  TP  SEGMENT THRESHOLD -
00010 PART-NBR      06  019 008  U   N   001
00020 SHORT-DESC   08  019 031  P   N   002
00030 RB STARTS ROW 12  000 000  P   B   003
55    ERR-MSG      20  002 074  P   B   004
00050 PAGING       21  002 008  P   D   005

```

```

COMMAND ==>
ORDER  NAME      ROW  COL  LEN  TP  SEGMENT THRESHOLD -
00010 PART-NBR      06  019 008  U   N   001
00020 SHORT-DESC   08  019 031  P   N   002
00030 RB STARTS ROW 12  000 000  P   B   003
00040 PAGING       21  002 008  P   D   005
00050 ERR-MSG      20  002 074  P   B   004

```

MOCK

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Compatibility: Batch environments

Description: Specify the report mock-up name in your report program.

Syntax: -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
 MOCK *mockupname*

Comment: Mockupname is the name entered in the Reports field associated with your program in the Application Painter and painted in the Report Painter.

Example:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
RED    STOCK-REPORT
        CONTROLS ARE FINAL WS-LOCATION-CODE
        PAGE LIMIT IS 50
        FIRST DETAIL 10
        LAST DETAIL 40
        FOOTING 47.
MOCK   STCKPRT
01     TYPE IS REPORT HEADING NEXT GROUP
        NEXT PAGE LINE 20.
        MOCKUP LINES 1 THRU 6
        SOURCE WS-DATE
```

MOCKUP LINES

Category: Report Writer statement (see *Report Writer Structures* and the *APS User's Guide* chapter *Create Reports with Report Writer*.)

Compatibility: Batch environments

Description: In your report program, specify the line numbers in the report mock-up that correspond to the detail lines, headers, and footers.

Syntax: `MOCKUP|M LINE|LINES linenumber1 [THRU linenumberN]`

- Comments:**
- Refer to all extra lines in a report mock-up with a `MOCKUP` statement.
 - The maximum number of report mock-up lines is 200.

Example: Specify that lines 1 through 8 in the report mock-up contain the heading text.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
01     TYPE IS REPORT HEADING LINE 20 NEXT GROUP
        NEXT PAGE.
        MOCKUP LINES 1 THRU 8
```

Mock-Up Report (RP01)

Category: APS-generated report (see *Application Reports*)

Description: The Mock-Up Report contains a representation of a report mock-up as painted in the Report Painter. When produced from the Report Generator, the report includes line numbers. This report documents a key aspect of an application for end users to review or for developers to use when maintaining or enhancing the application.

- Comments:**
- Produce the Mock-Up Report from the Report Generator, Painter Menu, or Application Painter.
 - You can set or change generation options for the report. To do so, access the Report Options screen in one of the following ways.
 - From the Report Generator screen, enter **4** in the **Option** field.
 - From any APS screen, enter **opt** in the **Command** or **Options** field. From the APS Options Menu that displays, enter option **4** in the **Options** field.

The Report Options screen displays. Specify **y(es)** to print line numbers or **n(o)** if you do not want to print them.

Example:

```
REPORT CODE: REPT                      APS ENTITY REPORT FACILITY          PAGE    1
                                         CLSAPS.CLS2                          01/18/92 15:01
```

```
REPORT CRITERIA: 2
```

```
ALL MEMBERS OF LIBRARY TYPE : RP
```

```
*****
```

LIBRARY TYPE	ENTITY NAME	STATUS	REMARKS
RP	COSTRPT	REPORTED	
RP	MERPT	REPORTED	

```
REPORT CODE: RP01                      APS APPLICATION PAINTER          PAGE    1
                                         APPLICATION DEFINITION REPORT    01/18/92 15:01
                                         CLSAPS.CLS2
```

```
SELECTION CRITERIA: COSTRPT
```

```
*****
REPORT: COSTRPT                        CREATED: 07/27/90
AUTHOR: CLSONE                          UPDATED: 07/27/90
*****
```

```
USER MANUFACTURING COMPANY
EXPENDITURES REPORT AS OF XXXXXXXX
```


that are turned off. APS generates the logic for all screens listed in the SCRNLIST call or the screen specified in the NTRY call.

- Comments:**
- During processing, APS returns the modified data to the program, and retains the unmodified data in temporary storage and returns it to the program after the modified data is returned.
 - A macro, also in APCICSIN, named \$TP-MDTOFF-BUILD-QUEUENAME builds the temporary storage queue name used to store the screen records. You can change this macro to build a name that conforms to your site standards.
 - Also contained in APCICSIN are the following error processing macros that execute when there is an error in reading, writing, or deleting your screen records in temporary storage.

```
$TP-MDTOFF-READQ-TS-ERROR
$TP-MDTOFF-WRITEQ-TS-ERROR
$TP-MDTOFF-DELETEQ-TS-ERROR
```

As shipped, these macros generate a CICS-SEND-TEXT call to display appropriate error messages, and a TERM to terminate the program. You can change these macros to conform to your shop standards.

MSG-SW

Category: Data communication call (see *Data Communication Calls*)

Compatibility: IMS DC and ISPF Prototyping targets

Description: Send a data or a message to the specified program trancode or subprogram.

Syntax: IMS DC

```
[TP-]MSG-SW trancode|programname|dataname [errorpara]
... [screenname|recordname]
... [keyword[+keyword]...]
```

ISPF Prototyping

[TP-]MSG-SW *trancode*|*programname*|*dataname* [*errorpara*]
 ... [*screenname*]

Parameters:	<i>dataname</i>	Name of data element.
	<i>errorpara</i>	User-defined error routine to perform when an abnormal condition occurs. <i>Errorpara</i> is positional; if omitted, code an asterisk (*) in its place.
	<i>keyword</i>	Valid keywords are:
	NOALTRESP	Default. Use an IO PCB, not an alternate response IO PCB to send a response to the terminal.
	ALTRESP	Use an alternate response IO PCB to send a response to the terminal.
	NOCONT	Default. Control returns to top of the Procedure Division of the sending program.
	CONT	Control returns to the instruction following call execution, that is, the next statement after the MSG-SW.
	CONTCOND	TP-CONTCOND determines if control passes to the next instruction or returns to the top of the program.
	NOEXPRESS	Default. Do not send a message for abnormal program termination.
	EXPRESS	Send a message at program termination.
	NOENDCONV	Default. Do not blank out the TRancode in the SPA.
	ENDCONV	Blank out the TRancode in the SPA.

	SCREEN	Default. Input is an APS-painted screen. Multisegment screens are not supported.
	RECORD	Input is <i>recordname</i> . See also "Comments" below.
	NOPURG	Send all messages to the same destination as one multi-segmented message. Default with NOEXPRESS keyword.
	PURG	After inserting the message, send it as one single-segmented message. Default with EXPRESS keyword.
<i>recordname</i>		User-defined I/O area in Working-Storage. See also "Comments" below.
<i>screenname</i>		Screen name; must be literal (maximum 8 characters).
<i>trancode programname</i>		Literal name of destination program or transaction code identifying receiving program; can be a literal (maximum 8 characters) or a COBOL data name (minimum 9 characters).

- Comments:**
- Allow the program to send invalid or empty screen values by specifying NORETRY with the NTRY keyword; use generated flags to test whether input data is valid.
 - Use the PURG and NOPURG keywords to control how APS sends messages. When issuing multiple inserts to the same destination before reaching a SYNC point, IMS sends the messages as one multi-segmented message. If you send the message via an Express PCB, APS issues a PURG after the insert and sends it as one single-segmented message.
 - Omitting screenname or recordname sends a message with a transaction code to the new program.
 - Making both the sending and receiving program conversational passes data in the Commarea (SPA) to the new program (see *TP-COMMAREA*).
 - MSG-SW lets a program send a message to itself. You might use this feature to break a long process into several small pieces.

- When coding RECORD and recordname in conversational programs, and recordname differs from your application trancode, code
SYM1 % &recordname-TRANCODE = "trancode"
- Remember to define an alternate IO PCB in the program specification block (PSB) and specify MODIFY=YES (see *Program Control Blocks, IO*).

Examples: Send messages to another program; perform error routine when error occurs.

```
MSG-SW trancode PROCESS-ERRORS
```

Send messages and pass a screen record. Note that the asterisk (*) replaces the positional parameter *errorpara*.

```
MSG-SW trancode * SCRA
```

Send messages and pass a COBOL record.

```
MSG-SW trancode * WS-CUSTOMER RECORD
```

NTRY

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Description: Generate a program template that fully defines all parts of your program except for the procedural code that you supply.

NTRY generates a program template that defines:

- The Identification Division, based on your Application Painter specifications
- The Environment Division, based on your Application Painter specifications
- The Data Division, including the following Working-Storage and Linkage Section structures:
 - Your database record or file definitions, based on your imported subschema

- Your screen field data structures, based on your Screen Painter specifications
- CICS EIBRCODE and DFHCOMMAREA structures
- Your IMS PCB mask, including I/O and database PCBs, based on your imported subschema
- An APS data structure for passing data among programs, known as a Commarea; the Commarea appears in either Working-Storage or the Linkage Section, depending on your DC target
- PF key definitions, based on your specified DC target
- Flags required by APS
- Portions of the Procedure Division, including:
 - A housekeeping routine, to initialize Working-Storage fields, flags, and counters
 - Program invocation logic, to initialize your program when it is invoked by a transaction ID, a screen, or another program, based on your specified DC target
 - Logic to send the program window or screen to the end user's monitor

In CICS programs, NTRY also generates code to:

- Obtain addressability to Linkage Section areas passed from other programs via addresses in the program Commarea.
- Receive correct map if transaction is screen-invoked.

In ISPF Dialog programs, NTRY also generates code to display screens.

In IMS programs, NTRY also generates code to:

- Specify input as a screen or a user-defined I/O area in Working-Storage.
- Generate code to receive the specified screen.
- Read SPA if the program is conversational.

Syntax: CICS

```
NTRY|ENTR screenname(mapsetname) ]
... [ errorpara ]
... [ RETRY|NORETRY ]
```

IMS DC**Format 1:**

```
NTRY|ENTR
```

Format 2:

```
NTRY|ENTR screenname
... [ errorpara ]
... [ RETRY|NORETRY ]
```

Format 3:

```
NTRY|ENTR recordname
... [ errorpara ]
... [ RETRY|NORETRY ]
... *RECORD
```

ISPF Dialog

```
NTRY|ENTR screenname
... [ errorpara ]
... [ RETRY|NORETRY ]
... [ CANCEL|RETURN ]
... [ dataareas ]
```

ISPF Prototyping

```
NTRY|ENTR screenname
... [ RETRY|NORETRY ]
... [ CANCEL|RETURN ]
... [ dataareas ]
```

Parameters:	CANCEL	Default. Generate CONTROL ERRORS CANCEL (return control to ISPF when dialog error of return code 12 or higher).
	<i>dataareas</i>	Generate the PROCEDURE DIVISION USING statement. List user-defined data areas only, not records defined by TP-COMMAREA, TP-LINKAGE, and SCRNLIST, which automatically generate other data areas. APS generates data areas in the following order.

Data areas listed with NTRY.

- *TP-COMMAREA* record
- *TP-LINKAGE* records
- *SCRNLIST* screen records

<i>errorpara</i>	User-defined error routine to perform when an abnormal condition occurs. <i>Errorpara</i> is positional; if omitted, code an asterisk (*) in its place.
<i>mapsetname</i>	Mapset containing the screen(s) the program receives; must be a literal (maximum 7 characters). See also Comments: for CICS below.
NORETRY	Accept invalid data in screen fields with assigned edits. See also Comments: below.
*RECORD	Code when using recordname instead of screenname as input.
<i>recordname</i>	<p>User-defined I/O area in Working-Storage, long enough for any input message that can be received. The area receives the data returned from the GU to the program control block.</p> <p>For a conversational program, the program reads the ASPA with a GU, and the input message with a GN. It does not expect a multisegment input message. PF keys are not generated; you must code them.</p>
RETRY	Default. Accept only valid data in screen fields with assigned edits; otherwise, return screen to the terminal user for correction. See also "Comments" below.
RETURN	Generate CONTROL ERRORS RETURN to return control to program when dialog error occurs. See also Comments: below.
<i>screenname</i>	Screen name; value must be literal (maximum 8 characters).

Comments: Code an NTRY, PROC, or OPT statement for each program.

- To generate a batch program template with a PROCEDURE DIVISION USING statement, code *PROC*.
- The generated \$TP-ENTRY generates a program template to send the appropriate window depending on the program invocation mode. To suppress the window and program invocation logic from your program template, code OPT PROG (*OPT*).

CICS

- Normally, if received data fails to pass editing, APS returns the screen, highlighting errors and displaying error messages in the SYSMMSG field. Application logic executes after the received data passes editing.

It is possible, however, to accept screens with data that does not pass editing. When NORETRY is coded, the application logic executes regardless. A set of runtime flags allow application logic to determine the success or failure of editing.

APS-EDITS-PASSED	88-level flag set to true when data for all edited fields is valid. Code IF APS-EDITS-PASSED to test whether the edit is OK.
<i>screenname-fieldname</i> -FLAG	Indicator flag set to spaces when data for a specified field valid. Code IF <i>screenname-fieldname</i> -FLAG = SPACES to test whether the data is valid.
<i>screenname-fieldname</i> -INPT	APS-generated data field; contains data exactly as entered if the field failed to pass editing.
APS-MSG-IO-ERROR	Applies to IMS DC only. 88-level flag set to true when a program sends a user-defined I/O error message. Code IF APS-MSG-IO-ERROR to test if message sent.
APS-MSG-EDIT-ERROR	88-level flag set to true when a program sends a user-defined edit error message. Code IF APS-MSG-EDIT-ERROR to test if the message is sent.

- When *mapsetname* is not specified, APS looks for it in the Screen Generation Parameters screen information; otherwise, APS generates a default mapset name, as follows.

Screen Name Length	Generated Mapset Name
4 characters	<i>screenname</i> SET
5 characters	<i>screenname</i> \$
6 characters	<i>screenname</i> \$
7 characters	Last character of <i>screenname</i> changes to \$
8 characters	Truncates eighth character of <i>screenname</i> ; seventh character changes to \$

- You can write macros, using the following names, to customize NTRY processing.

User Macro Name	Where Included
TP-ENTRY-EXIT-1	Bottom of APS-HOUSEKEEPING-PARA
TP-ENTRY-EXIT-2	After APS-HOUSEKEEPING-PARA

IMS DC

- Normally if received data fails to pass editing, APS returns the screen, highlighting errors and displaying error messages in the SYSMMSG field. Application logic executes after the received data passes editing.

It is possible, however, to accept screens with data that does not pass editing. When NORETRY is coded, the application logic executes regardless. A set of runtime flags allow application logic to determine the success or failure of editing.

APS-EDITS-PASSED	88-level flag set to true when data for all edited fields is valid. Code IF APS-EDITS-PASSED to test whether the edit is OK.
<i>screenname-fieldname</i> -FLAG	Indicator flag set to spaces when data for a specified field valid. Code IF <i>screenname-fieldname</i> -FLAG = SPACES to test whether the data is valid.

<i>screenname-fieldname</i> -INPT	APS-generated data field; contains data exactly as entered if the field failed to pass editing.
APS-MSG-IO-ERROR	Applies to IMS DC only. 88-level flag set to true when a program sends a user-defined I/O error message. Code IF APS-MSG-IO-ERROR to test if message sent.
APS-MSG-EDIT-ERROR	88-level flag set to true when a program sends a user-defined edit error message. Code IF APS-MSG-EDIT-ERROR to test whether if message sent.

- A serially reusable IMS online program processes multiple input messages in a single execution. After the program processes one input message and sends a response, it reads and processes another input message and terminates when there are no more input messages.

The APS/IMS DC Generator creates programs consistent with this practice. Termination DC calls (SEND, MSG-SW, TERM) do not terminate a program, but instead return control to NTRY-generated logic to receive a new input message.

Because a program may process multiple messages in a single execution, you program should perform any Working-Storage initialization in the Procedure Division, rather than in Working-Storage using the VALUE clause. You can still use the VALUE clause to initialize Working-Storage constants that are never changed during execution.

- You can write macros, using the following names, to customize NTRY processing.

User Macro Name	Where Included
TP-ENTRY-EXIT-1	Bottom of APS-HOUSEKEEPING-PARA
TP-ENTRY-EXIT-2	After APS-HOUSEKEEPING-PARA
TP-ENTRY-EXIT-2S	After screen message is sent
TP-ENTRY-EXIT-3	At end of \$TP-ENTRY macro

ISPF Dialog

- Normally if received data fails to pass editing, APS returns the screen, highlighting errors and displaying error messages in the SYSMMSG field. Application logic executes after the received data passes editing.

It is possible, however, to accept screens with data that does not pass editing. When NORETRY is coded, the application logic executes regardless. A set of runtime flags allow application logic to determine the success or failure of editing.

APS-EDITS-PASSED	88-level flag set to true when data for all edited fields is valid. Code IF APS-EDITS-PASSED to test whether the edit is OK.
<i>screenname-fieldname-FLAG</i>	Indicator flag set to spaces when data for a specified field valid. Code IF <i>screenname-fieldname-FLAG</i> = SPACES to test whether the data is valid.
<i>screenname-fieldname-INPT</i>	APS-generated data field; contains data exactly as entered if the field failed to pass editing.
APS-MSG-IO-ERROR	Applies to IMS DC only. 88-level flag set to true when a program sends a user-defined I/O error message. Code IF APS-MSG-IO-ERROR to test if message sent.
APS-MSG-EDIT-ERROR	88-level flag set to true when a program sends a user-defined edit error message. Code IF APS-MSG-EDIT-ERROR to test whether if message sent.

- Code logic in errorpara to process all ISPEXEC display service return codes with a value of 12 or higher.
- Provide errorpara to handle dialog errors when coding RETURN.

Examples: Generate code to receive screen SCRA when the program is screen-invoked. Specify that the program can receive a screen that contains invalid data. Omit the error paragraph.

```
-KYWD- 12--*--20---*-----30-----*---40---*-----50---*-----60
NTRY   SCRA * NORETRY
```

Receive multiple screens and include an error routine. Note that when you use SCRNLIST, you do not specify any screens in the NTRY statement.

```
-KYWD- 12--*--20---*-----30-----*---40---*-----50---*-----60
SYM1   SCRNLIST C1ORDR C2ORDR C3ORDR
NTRY   * PROCESS-ERRORS
```

CICS

Generate code to receive screen CUSTORDR in mapset CUSTOR\$ for a screen-invoked program. Perform MAP-ERROR-PARA when an exceptional condition occurs on the RECEIVE MAP.

```
-KYWD- 12--*--20---*-----30-----*---40---*-----50---*-----60
NTRY   CUSTODR(CUSTOR$) MAP-ERROR-PARA
```

IMS DC

Receive a message into a COBOL record I/O area.

```
-KYWD- 12--*--20---*-----30-----*---40---*-----50---*-----60
NTRY   WS-CUSTOMER * *RECORD
```

DLG

Generate code to display screen SCRA. Perform PROCESS-ISPF-ERRORS when return codes from services are greater than 12. Generate CONTROL ERRORS RETURN code so the program can control all error processing.

```
-KYWD- 12--*--20---*-----30-----*---40---*-----50---*-----60
NTRY   SCRA PROCESS-ISPF-ERRORS RETURN
```

ISPF Dialog

Generate code to display screen SCRA. Perform PROCESS-ISPF-ERRORS when return codes from services are greater than 12. Generate CONTROL ERRORS RETURN code so the program can control all error processing.

```
-KYWD- 12--*--20---*-----30-----*---40---*-----50---*-----60
NTRY   SCRA PROCESS-ISPF-ERRORS RETURN
```

NULL Indicator Field

Compatibility: SQL target

Description: Use a null indicator variable to indicate whether the associated host variable has been assigned a null value.

Syntax: APS/SQL generates a null indicator variable in Working-Storage, defined as follows.

```
01  IND-cursorname|IND-recordname
    05  IND-column
```

- Comments:**
- When using indicator variables, prefix them with the 01 level name, because APS can generate duplicate names with different 01 levels.
 - When column contains an underscore, it changes to a hyphen at program generation.
 - Normally, the IND-cursorname structure references a cursor set. To override this structure when generating indicator variables, set the &D2-USE-CURSOR-IND flag to NO in the APS CNTL member APDB2IN. This generates the indicator variables with the IND-recordname structure.

OCCURS

Category: Data Structure Painter construct (see *Data Structures*)

Description: Code OCCURS clauses in your data structures.

Syntax:

```
dataname(OCCURSclosure) [TIMES]
[... DO|ODO dataname]
[... IX|IB|IXB|IXBY dataname]
[... ASCENDING KEY IS dataname]
[... DESCENDING KEY IS dataname]
... PICformat
```

Shorthand syntax for the dimensions of a table in an OCCURS clause.

OCCURS Format	Generated Code
<i>(number)</i>	OCCURS <i>number</i>
<i>(number)</i> TIMES	OCCURS <i>number</i> TIMES
<i>(number1-number2)</i>	OCCURS <i>number1</i> TO <i>number2</i>
<i>(number1 TO number2)</i>	OCCURS <i>number1</i> TO <i>number2</i>
<i>(&variable)</i>	OCCURS <i>&variable</i>
<i>(&variable1 TO &variable2)</i>	OCCURS <i>&variable1</i> TO <i>&variable2</i>

Parameters: DO|ODO Generates DEPENDING ON
 ... IX|IXB|IXBY|IB Generates INDEXED BY

Comment: Always code an INDEXED BY clause on a continuation line.

Examples: Data Structure Painter format:

```
-LINE- ----- Data Structure Painter -----
000001      EXDS-TABLE
000002          EXDS-TABLE-3  (1-99)
000003          ... ODO EXDS-TABLE-3-SIZE
000004          ... IXB EXDS-INDEX
000005          ... X20
```

Generated COBOL code:

```
01  EXDS-TABLE.
    05  EXDS-TABLE-3 OCCURS 1 TO 99
        DEPENDING ON
        EXDS-TABLE-3-SIZE
        INDEXED BY EXDS-INDEX
        PIC X(20).
```

Data Structure Painter format:

```
-LINE- ----- Data Structure Painter -----
000001      TYPE-DESC-RATE-CONSTANTS X90
000002      TYPE-DESC-RATE-TABLE-REDEF R
000003          TYPE-DESC-RATE-TABLE (30)
000004          ... ASCENDING KEY IS TYPE-CODE
000005          ... IXB DATA-INDEX
000006          TYPE-CODE X
000007          DESC-CODE X
000008          RATE-CODE X
```

Generated COBOL code:

```

01 TYPE-DESC-RATE-CONSTANTS    PIC X(90).
01 TYPE-DESC-RATE-TABLE-REDEF  REDEFINES
    TYPE-DESC-RATE-CONSTANTS.
    05 TYPE-DESC-RATE-TABLE    OCCURS 30
        ASCENDING KEY IS TYPE-CODE
        INDEXED BY DATA-INDEX.
        10 TYPE-CODE            PIC X.
        10 DESC-CODE           PIC X.
        10 RATE-CODE           PIC X.

```

Data Structure Painter format:

```

-LINE- ----- Data Structure Painter -----
000001   LOAN-RATE-TABLE
000002       LOAN-RATE-ROW OCCURS 10 TIMES
000003           ... ASCENDING KEY IS TYPE-CODE
000004           ... IXB DATA-INDEX
000005               TYPE-CODE PIC X(05)

```

Generated COBOL code:

```

01 LOAN-RATE-TABLE.
    05 LOAN-RATE-ROW    OCCURS 10 TIMES
        ASCENDING KEY IS TYPE-CODE
        INDEXED BY DATA-INDEX.
        10 TYPE-CODE    PIC X(05).

```

OPT

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Compatibility: Programs created in Program Painter

Description: Suppress the mainline program section generated by the NTRY or PROC keyword, in order to supply your own screen and program invocation logic. Use OPT only in programs with screens.

Syntax: -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
 NTRY |
 PROC
 OPT PROG

- Comment**
- You can use PROGRAM, PROGRAMMER, PCONV, or PCONVERT in place of PROG.
 - Code an NTRY, PROC, or OPT statement for each program.
 - To generate a batch program template with a PROCEDURE DIVISION USING statement, code *PROC*.
 - To generate a program template that fully defines all parts of your program except for the procedural code that you supply, code *NTRY*.

OVERPRINT

Category: Report Writer statement (see *Report Writer Structures* and the *APS User's Guide* chapter *Create Reports with Report Writer*.)

Compatibility: Batch environments

Description: Print one line on top of the other without advancing the line printer. Use this feature to create bold text or to underscore text.

Syntax: OVERPRINT|O WHEN '*characterstring*' AT COLUMN *integer*

Parameters: *characterstring* Line or text to be overprinted; must be identical to *characterstring* in mock-up; delimit with single or double quotation marks.

integer Starting column number of *characterstring* on the mock-up.

Comment: In the mock-up, the two lines of text that print on one line of the page must be consecutive. On the first line enter a text string. On the second line enter the text that prints over the first line, and to the right of this text enter a unique characterstring to identify the line. When the report prints, blanks replace *characterstring* in the mock-up.

Examples: Underscore text within a page heading. Below are lines 7 and 8 of a mock-up. The identifying character string, NO ADVANCING, begins in column 70 of the second line.

```
XXXXXXXX
_____
NO ADVANCING
```

The OVERPRINT clause:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
01     TYPE IS PAGE HEADING.
      MOCKUP LINES 7 THRU 9
      OVERPRINT WHEN 'NO ADVANCING' AT COLUMN 70
```

The printed result:

```
XXXXXXXX
```

Print text within the report heading of a mock-up twice, to appear as bold type. The identifying character string begins in column 70.

```
WIDGETS
WIDGETS
DITTO
```

The OVERPRINT statement:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
01     TYPE IS REPORT HEADING LINE 20 NEXT GROUP NEXT PAGE.
      MOCKUP LINES 1 THRU 6
      OVERPRINT WHEN 'DITTO' AT COLUMN 70
```

The printed result:

```
WIDGETS
```

PAGE LIMIT

Category: Report Writer clause (see *Report Writer Structures* and the *APS User's Guide* chapter *Create Reports with Report Writer*.)

Compatibility: Batch environments

Description: Define the page length and the vertical subdivisions of a printed page.

Syntax: PAGE LIMIT IS|ARE *number* [LINE|LINES]
 [FIRST DETAIL *linenumber*]
 [LAST DETAIL *linenumber*]
 [FOOTING *linenumber*] [.]

Parameters:

<i>number</i> LINE LINES	Number of lines on each report page. Number cannot exceed 3 digits and must be greater than or equal to the FOOTING <i>linenumber</i> .
FIRST DETAIL <i>linenumber</i>	First detail line. Print control break heading and report body detail lines beginning on <i>linenumber</i> . Print REPORT and PAGE HEADING groups before <i>linenumber</i> .
LAST DETAIL <i>linenumber</i>	Line number of the last report body detail line. Print CONTROL FOOTING, PAGE FOOTING, and REPORT FOOTING lines after this number. <i>Linenumber</i> must be greater than FIRST DETAIL <i>linenumber</i> .
FOOTING <i>linenumber</i>	Last line number of the last control footing report group. Print PAGE FOOTING and REPORT FOOTING lines after this number. <i>Linenumber</i> must be greater than or equal to LAST DETAIL <i>linenumber</i> .

- Comments:**
- Code PAGE LIMIT before FIRST DETAIL, LAST DETAIL, or FOOTING.
 - Code FIRST DETAIL, LAST DETAIL, and FOOTING in any order.
 - Each report group should fit on one page; Report Writer never splits a report group across page boundaries.
 - Coding PAGE LIMIT assumes default values.
 - Omitting FIRST DETAIL assigns the heading line value to *linenumber*.
 - Omitting either LAST DETAIL or FOOTING assigns the PAGE number value to them.
 - Coding FOOTING without LAST DETAIL assigns the FOOTING *linenumber* to LAST DETAIL *linenumber*.
 - Coding LAST DETAIL without FOOTING assigns the LAST DETAIL *linenumber* to FOOTING *linenumber*.

- Omitting PAGE LIMIT generates a single-page report of indefinite length without page and line counter registers.

Example:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
RED    STOCK-REPORT
        CONTROLS ARE FINAL WS-LOCATION-CODE
        PAGE LIMIT IS 50
        FIRST DETAIL 10
        LAST DETAIL 40
        FOOTING 47.
```

Panel Options, ISPF Dialog

Compatibility: ISPF Dialog target

Category: Screen Painter feature

Description: Generate native panel definition statements.

Procedure: To generate native panel definition statements, follow these steps.

- 1 From any screen in the Screen Painter, select Actions ISPF Panel Options from the action bar or enter *is* in the Command field. The ISPF Panel Options screen displays.
- 2 Complete the screen fields as follows:

Field	Description and Values
Command Field	Enter screen field name for the panel command field that allows end users to enter ISPF commands and prevents truncation errors when they use PF keys. Default is the first unprotected I/O field on the screen. This option generates the native)BODY CMD(variable) statement.

Field	Description and Values
Long Message Short Message	<p>Enter <i>sysmsg</i>, if specified on Screen Generation Parameters screen, or a screen field name for the panel long and short message fields. These fields allow the program to move literal messages to the screen.</p> <p>These options generate the native)BODY LMSG(variable) statement for the long message and the)BODY SMSG(variable) statement for the short message.</p>
Help Panel	<p>Enter the name of the panel to display, if the end user requests help.</p> <p>This option generates the HELP = <i>panelname</i> statement.</p>
Pfkey Option	<p><i>P</i> Program controls PF key processing. APS saves the end user's original PF key values and replaces them with literals not recognized by ISPF, so that you can control the PF key usage. When the program terminates, the original PF key values are restored. See <i>PF Key Values</i>.</p> <p><i>I</i> Default. ISPF controls PF key processing.</p>

- 3 To save your selections and exit this screen, press F3 or enter **end** in the **Command** field. To exit without saving your selections, enter **cancel** in the **Command** field.

PARA and Paragraphs

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Description: Indicate a paragraph in your program code. A paragraph is a Procedure Division routine that you write and perform specifically for one

program. Use paragraphs to perform the following, depending on which APS tool you use.

Use paragraphs in...	To perform...
Online Express	Custom actions for events; Custom routines at window events
Specification Editor	Custom routines in the Procedure Division
Program Painter	Custom routines in the Procedure Division

Syntax: Format 1:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
      PARA  paragraphname [SECTION.]
           paragraphcode
```

Format 2:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60--
      .
      .
      PERFORM paragraphname [(arguments)]
      .
      .
      .
      PARA  paragraphname [(+|-arguments)]
           statement
      .
      .
      .
      PERFORM subparagraphname [(arguments)]
      PARA  subparagraphname [(+|-arguments)]
           statement
      .
      .
      .
      WS
      01    group-level-data-item
           05 elementary-data-item
      .
      .
      .
```

- Comments:**
- Use Format 1 with SECTION to code a SORT procedure.
 - The keyword PARA, followed by a paragraph name with an optional list of arguments, denotes a paragraph. Within a paragraph,

indentation determines the exact positioning of its logic statements, or statement blocks.

- A paragraph ends with the next appearance of any Procedure Division keyword, except the comment keyword (*/**); there is no need to code an *END-paragraphname*.
- Code at least one PERFORM statement before any PARA keywords in your program. Generally, you code a PERFORM statement that performs your main logic paragraph.

Paragraph Rules:

Rules for coding paragraphs in Online Express:

- A paragraph can consist of a main paragraph, other paragraphs that the main paragraph performs, and Data Division source code for the paragraphs.
- For each paragraph, enter the PARA keyword in the KYWD column and the paragraph name in column 12 on the same line. On the following lines, enter COBOL, COBOL/2, or S-COBOL paragraph statements. Do not use any other APS keywords in paragraphs.
- After all paragraphs, use APS Data Division keywords to define data items that the paragraphs reference.

Rules for coding paragraphs in the Specification Editor and Program Painter:

- A paragraph can perform other paragraphs.
- For each paragraph, enter the PARA keyword in the KYWD column and the paragraph name in column 12 on the same line. On the following lines, enter COBOL, COBOL/2, or S-COBOL paragraph statements. Do not use any other APS keywords in paragraphs.

Anywhere in the program, use APS Data Division keywords to define any data items that the paragraphs reference.

PERFORM

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Depart from the normal program execution sequence to execute a particular paragraph or section and then return control to the statement immediately following the PERFORM statement.

Syntax: Format 1, perform a paragraph:

```
PERFORM paragraphname
```

Format 2, perform a paragraph and pass arguments:

```
PERFORM paragraphname (actualargument1[, ]
... actualargument2[, ] ... actualargumentN)
.
.
.
PARA paragraphname ([+|-]formalargument1[, ]
... [+|-]formalargument2[, ] ... [+|-]formalargumentN)
```

Parameters: +|-

A plus (+) or minus (-) sign preceding a formal argument passes values between actual arguments and formal arguments, as follows.

- With a plus sign (+), PERFORM passes the actualargument value to formalargument after the paragraph executes. The program does not return the formalargument value to *actualargument*.
- With a minus sign (-), PERFORM passes the formalargument value to the actualargument after the paragraph executes.
- If there is no plus or minus sign, the PERFORM passes the actualargument to its corresponding formalargument. After the paragraph executes, PERFORM passes the formalargument back to its corresponding actualargument.

<i>actualargument</i>	Values you send to or receive from a formalargument in the paragraph; can be literals, identifiers, arithmetic expressions, variables, or index names.
<i>formalargument</i>	Values you receive from or send to an actualargument in the PERFORM statement.
<i>paragraphname</i>	Name of paragraph to perform.

- Comments:**
- When continuing a list of arguments onto one or more other lines, do not break an argument.
 - The number of actual arguments must equal the number of formal argument names and be in the same order.

Examples:

```
PERFORM PARA-1( 'ABC' , ABC-FLD, RESULT)
      .
      .
      .
      PARA-1( +ARG1, +ARG2, -'Y')
```

Generates:

```
MOVE 'ABC' TO ARG-1
MOVE ABC-FLD TO ARG2
PERFORM PARA-1 THRU .....
MOVE 'Y' TO RESULT
```

Perform GET-SALARY (line 5010); pass the literal 20 to HOURS-WORKED and the value of HOURLY-RATE to PAY-RATE before calculating TOT-PAY (line 5020). After the GET-SALARY paragraph executes, pass the value of PAY-RATE to HOURLY-RATE and the value of TOT-PAY to WEEKLY-PAY.

```
-LINE- -KYWD- 12-*----20---*----30---*----40---*----50---*---
000100  NTRY
      .
      .
      .
003010          IF CLASS = 'HALF-TIME'
003020              PERFORM GET-SALARY( 20,
003021                  ... HOURLY-RATE, WEEKLY-PAY)
003030          ELSE
003040              PERFORM GET-SALARY( 40,
003041                  ... HOURLY-RATE, WEEKLY-PAY)
003050          WEEKLY-PAY = WEEKLY-PAY * TAX
003060          COMPUTE MEDICAL-DEDUC =
003061          ... HOURLY-RATE * 1.50
```

```

.
.
.
005010  PARA    GET-SALARY( +HOURS-WORKED,
005011          ...  PAY-RATE, -TOT-PAY)
005020          TOT-PAY = HOURS-WORKED * PAY-RATE
.
.
.

```

PF Key Values

Category: Data communications feature (see also *Data Communication Calls*)

Compatibility: CICS, IMS DC, and ISPF Dialog targets

Description: Use the PF key 88-levels generated by APS.

Syntax: CICS

```

PFKEY-FIELD PIC X(01).
88  ENTER-KEY VALUE ''' . (single quotation mark)
88  CLEAR-KEY VALUE '_'.
88  PEN VALUE '=' .
88  OPID VALUE 'W' .
88  MSRE VALUE 'X' .
88  STRF VALUE 'H' .
88  TRIG VALUE '"'. (double quotation mark)
88  PA1 VALUE '%' .
88  PA2 VALUE '>' .
88  PA3 VALUE ', ' .
88  PF0 VALUE ''' . (single quotation mark)
88  PF00 VALUE ''' . (single quotation mark)
88  PF1 VALUE '1' .
88  PF2 VALUE '2' .
88  PF3 VALUE '3' .
88  PF4 VALUE '4' .
88  PF5 VALUE '5' .
88  PF6 VALUE '6' .
88  PF7 VALUE '7' .
88  PF8 VALUE '8' .
88  PF9 VALUE '9' .

```

```

88 PF1 VALUE ':' .
88 PF11 VALUE '#' .
88 PF12 VALUE '@' .
88 PF13 VALUE 'A' .
88 PF14 VALUE 'B' .
88 PF15 VALUE 'C' .
88 PF16 VALUE 'D' .
88 PF17 VALUE 'E' .
88 PF18 VALUE 'F' .
88 PF19 VALUE 'G' .
88 PF20 VALUE 'H' .
88 PF21 VALUE 'I' .
88 PF22 VALUE '[' .
88 PF23 VALUE '.' .
88 PF24 VALUE '<' .

```

IMS DC

APS assigns character values for the 24 PF keys and the ENTER key to APS-painted screens. NTRY generates logic to place this portion of the input message in a field with 88-level condition names to facilitate program testing for PF and ENTER keys.

```

TP-PF-KEY          PIC X(132) .
88 PF0             VALUE ' ' .
88 PF00            VALUE ' ' .
88 PF1             VALUE '1' .
88 PF01            VALUE '1' .
88 PF2             VALUE '2' .
88 PF02            VALUE '2' .
88 PF3             VALUE '3' .
88 PF03            VALUE '3' .
88 PF4             VALUE '4' .
88 PF04            VALUE '4' .
88 PF5             VALUE '5' .
88 PF05            VALUE '5' .
88 PF6             VALUE '6' .
88 PF06            VALUE '6' .
88 PF7             VALUE '7' .
88 PF07            VALUE '7' .
88 PF8             VALUE '8' .
88 PF08            VALUE '8' .
88 PF9             VALUE '9' .
88 PF09            VALUE '9' .
88 PF10            VALUE 'A' .
88 PF11            VALUE 'B' .
88 PF12            VALUE 'C' .
88 PF13            VALUE 'D' .

```

```

88 PF14          VALUE 'E'.
88 PF15          VALUE 'F'.
88 PF16          VALUE 'G'.
88 PF17          VALUE 'H'.
88 PF18          VALUE 'I'.
88 PF19          VALUE 'J'.
88 PF20          VALUE 'K'.
88 PF21          VALUE 'L'.
88 PF22          VALUE 'M'.
88 PF23          VALUE 'N'.
88 PF24          VALUE 'O'.
88 ENTER-KEY    VALUE ' '.
88 NO-KEY-USED  VALUE LOW-VALUES.

```

During screen painting, if you paint PF key values or use a PF key to supply all or part of the trancode value, you cannot use the above facility for PF key testing.

ISPF Dialog

```

TP-PF-KEY          PIC X(04).
88 ENTER-KEY      VALUE '    '.
88 PF1            VALUE 'PF01'.
88 PF01           VALUE 'PF01'.
88 PF2            VALUE 'PF02'.
88 PF02           VALUE 'PF02'.
88 PF3            VALUE 'PF03'.
88 PF03           VALUE 'PF03'.
88 PF4            VALUE 'PF04'.
88 PF04           VALUE 'PF04'.
88 PF5            VALUE 'PF05'.
88 PF05           VALUE 'PF05'.
88 PF6            VALUE 'PF06'.
88 PF06           VALUE 'PF06'.
88 PF7            VALUE 'PF07'.
88 PF07           VALUE 'PF07'.
88 PF8            VALUE 'PF08'.
88 PF08           VALUE 'PF08'.
88 PF9            VALUE 'PF09'.
88 PF09           VALUE 'PF09'.
88 PF10           VALUE 'PF10'.
88 PF11           VALUE 'PF11'.
88 PF12           VALUE 'PF12'.
88 PF13           VALUE 'PF13'.
88 PF14           VALUE 'PF14'.
88 PF15           VALUE 'PF15'.
88 PF16           VALUE 'PF16'.
88 PF17           VALUE 'PF17'.

```

```

88 PF18          VALUE 'PF18' .
88 PF19          VALUE 'PF19' .
88 PF20          VALUE 'PF20' .
88 PF21          VALUE 'PF21' .
88 PF22          VALUE 'PF22' .
88 PF23          VALUE 'PF23' .
88 PF24          VALUE 'PF24' .

```

To let your program control PF key values, specify P(rogram controlled) in the PFKEY Option field on the ISPF Panel Options screen. See *Panel Options, ISPF Dialog*.

Precompiler Options

Category: Application generation

Description: Define variations and special features for program precompilation.

- Procedure:**
- 1 From the APS Options Menu enter option **3** in the **Option** field. Alternatively, from any APS screen enter **opt 3** in the **Command** field. The Precompiler Options screen displays.
 - 2 Set options appropriate for your environment as described below.

Option	Description and Values
Apost	Override Quote.
	Yes Default. Use the apostrophe character to delimit non-numeric literals in your input source.
Quote	Override Apost.
	Yes Use the single quote character to delimit non-numeric literals in your input source.
	No Default
SCBtrace	Yes Activates the SAGE-TRACE-FLAG debugging facility.
RWT	Yes Default. Generate COBOL code from APS Report Writer statements. Specify with COBOL II compiler.
	No Pass Report Writer statements directly to the COBOL compiler.

Reference

Option	Description and Values	
<p>Note: For very large Report Writer programs, enter rwt=bigrwt in the APS Parm field on the Generator Options screen.</p>		
Lang	Indicate which type of source to process and which columns to process.	
	SCB=yes	Default. Processes APS specifications (S-COBOL) in columns 8-72; the symbol &07 in your code forces a character into column 7.
	COBOL=yes	Process COBOL source in columns 1-72.
	JCL=yes	Process JCL in columns 1-72. Useful for text-processing JCL and for controlling columns 1-6 of S-COBOL.
	Text=yes	Process any source in columns 1-80. APS considers all columns as text, and generates no sequence numbers. Automatically set XLATE=FMP. To override XLATE=FMP, enter XLATE=value in the APS Parm field. XLATE=value in the APS Parm field.
XLATE	Specify which processing step(s) that APS performs. You can stop processing at any of the steps listed below to help isolate the step at which errors occur. The steps are listed in the order in which APS executes them. All options except ALL are mutually exclusive.	
	ALL=yes	Default. Process all source code through all applicable processing steps and generates an error report; use when COBOL compile immediately follows in jobstream.
	FMP=yes	Stop processing after APS macros and user-defined Customization Facility macros are processed.
	RED=yes	Stop processing after report mock-ups are translated into IBM Report Writer source.
	RWT=yes	Stop processing after Report Section is translated into COBOL Working-Storage and S-COBOL.
MockupFMP	Yes	Scan lines in report mock-ups and processes the characters % \$ & and + as Customization Facility symbols.
	No	Default.
SUBR	Yes	Specify that the generated source is a subroutine program.
	No	Default. Specify that the generated source is a primary program.
Narrow	Yes	Default. Define 80 columns as the message report width.

Option	Description and Values	
Evalmess	No	Define 132 columns as the width.
	Yes	Generate messages that list evaluation bracket resolutions. Usually results in long listings.
Seq	No	Default.
		Specify the type of sequence numbers that APS generates. See also, Genident, Spaceident, Ident.
	COBOL=yes	Generate COBOL-style numbers in cols 1-6.
	Record=yes	Generate new numbers in columns 73-80, incrementing by 100 for each input record and by two for each generated record.
Syntax	Identifier=yes	Generate line numbers in columns 73-80; columns 73-74 contain 0.
		Specify which compiler to use.
	COBOLII=yes	Generate COBOL-II syntax.
Emark	S-COBOL=yes	Generate S-COBOL syntax.
		Generate a three-character string marking error and warning messages in the message report.
	Questions=yes	Default. Generate ???.
	Dollars=yes	Generate \$\$\$.
Genseq	3-Char String = <i>string</i>	Generate the string you specify.
		Override Spaceseq.
Spaceseq	Yes	Default. Generate sequence numbers in columns 1-6 for blank or out-of-sequence lines of source code and when new lines are generated.
		Override Genseq.
Genident	Yes	Generate spaces in columns 1-6; incompatible with Lang=Text.
		See also, Spaceident, Ident, Seq.
	Yes	Generate sequence numbers in columns 73-80 for blank or out of sequence source code lines and when new lines are generated.
	No	Default. Generate the last known contents of columns 73-80 when new lines are generated and passes identifiers as they exist in GENSRC.

Option	Description and Values	
Spaceident		See also, Genident, Ident, Seq.
	Yes	Generate spaces in columns 73-80. Incompatible with Lang=Text.
Main		Specify location of the main input source.
	MAININ=yes	Default. Read from file named by external name MAININ. Use this default unless using your own JCL.
	Instream=yes	Read source instream with the JCL that you provide.
	Member Name= <i>membername</i>	Read from the PDS or file name or source statement library designated by the external name SCELIB.
Ident		See also, Genident, Spaceident, Seq.
	Yes	Generate the internal program name in columns 73-80.
FMP	No	Default.
	Yes	Default. Process APS macros and user-defined Customization Facility macros.
Source	No	Use only with your own JCL skeleton.
	Yes	Print the main input source program, specified in the MAIN option, after the message report.
Gendirect	No	Default.
	Yes	Allow generation of nested IF statements in the COBOL source.
Gencomment	Yes	Generate replaced source statements as comments in the COBOL source.
	NO	Default.
Usernames	Yes	Generate the following prefix for APS-generated paragraphs: <i>paraname-</i>
	No	Default. Generate the following prefix for APS generated paragraphs: G--

Note: To generate any other prefix, enter the following in the APS Parm field on this screen. usernames=prefix

APS Parm Display all Precompiler options whose default values you override. These values also display in the APS Parm field on the Generator Options screen. APS saves the values you change on the APS Parm field on the Precompiler Option screen. APS does not save values that you change in the APS Parm field on the Generator Options screen.

Example:

```

OPTION ==> _
      APOST ==> YES      LANG=SCB ==> YES      XLATE=ALL ==>
      QUOTE ==>         COBOL ==>         FMP ==>
      SCBTRACE ==> NO   JCL ==>         RED ==>
      RWT ==> YES      TEXT ==>         RWT ==>
                                   SCB ==> YES
      MOCKUPFMP ==> NO
      SUBR ==> YES      SEQ=COBOL ==> YES   SYNTAX=COBOLII ==> YES
      NARROW ==> YES   RECORD ==>         S-COBOL ==>
      EVALMESS ==> NO   IDENTIFIER ==>
      GENSEQ ==> YES
      SPACESEQ ==> NO
      GENIDENT ==> NO   MAIN=MAININ ==> YES  EMARK=QUESTIONS ==> YES
      SPACEIDENT ==> NO INSTREAM ==>         DOLLARS ==>
                                   3-CHAR STRING ==>
      FMP ==> YES      MEMBER NAME ==>         IDENT=PGMID ==> NO
      SOURCE ==> NO
      GENDIRECT ==> YES
      GENCOMMENT ==> YES
      USERNAMES ==> NO
      APS Parm ==> XLATE=SCB

```

- Sequence and identify your generated source code lines as you prefer. For example, generate sequence numbers in columns 73 to 80, numbering lines by 100 for input records and by 2 for S-COBOL records.

```

GENIDENT ==> YES
SEQ=RECORD ==> YES

```

The generated lines are:

Input	S-COBOL
00000100	00000002
00000200	00000004
00000300	00000006

Generate sequence numbers in columns 73-80, with 00 in columns 73-74.

```

GENIDENT ==> YES
SEQ=IDENTIFIER ==> YES

```

The generated lines are:

Input	S-COBOL
00000001	00000001
00000002	00000002
00000003	00000003

- Stop APS processing at certain steps to help isolate the step at which errors occur. For example, process S-COBOL, the APS macros and user-defined Customization Facility macros and APS report mock-ups, and then stop.

```
XLATE=RED ===> YES
```

Process S-COBOL, and report mock-ups but do not process the APS macros or user-defined Customization Facility macros.

```
XLATE=RED ===> YES
FMP          ===> NO
```

Process only the APS macros and user-defined Customization Facility macros, columns 1-80. You must supply your own JCL to support this override.

```
XLATE=FMP ===> YES
LANG=TEXT ===> YES
```

- Specify the input source that APS reads if it resides in an external file other than MAININ. When doing so, you must supply your own JCL skeletons. For example, read the main input source from the file in MYLIB, called XYZ. You must supply your own SCELIB DD statement to describe MYLIB.

```
MAIN=MEMBER NAME ===> XYZ
```

- Override the default prefix that APS generates for paragraphs. For example, generate the prefix paraname- for all APS-generated paragraphs, rather than the default prefix G--.

```
USERNAMES ===> YES
```

Generate the prefix xyz- for all APS-generated paragraphs, rather than the default prefix G--.

```
APS Parm ===> USERNAMES=XYZ-
```

- Accept double quotation characters, rather than apostrophes, as the delimiters for non-numeric literals, and write an error message report for a 132-column output device.

```
QUOTE ===> YES
NARROW ===> NO
```

PROC

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Compatibility: Non-IMS batch programs

Description: Generate the batch program template, including a PROCEDURE DIVISION USING clause that enables a called program to receive data from the calling program.

Syntax: -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
PROC [variablename1 variablename2 ... variablenameN]

- Comments:**
- Code an NTRY, PROC, or OPT statement for each program.
 - The generated \$TP-ENTRY generates a program template to send the appropriate window depending on the program invocation mode. To suppress the window and program invocation logic from your program template, code OPT PROG (*OPT*).
 - To generate a program template to define all parts of your program except for the procedural code, code See *NTRY*.
 - Use PROC instead of NTRY in non-IMS batch programs called by other programs. For IMS programs, use NTRY.
 - Code Linkage Section data structures for the variables that the called program receives.

Example: Program Painter code:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
PROC   REC-2 REC-2 REC-3
```

Generated COBOL code:

```
PROCEDURE DIVISION USING REC-2 REC-2 REC-3
```

Program Control Blocks, IO

Category: Data communication feature (see also *Data Communication Calls*)

Compatibility: IMS DC target

Description: APS generates an IO PCB with the following format in the Linkage Section.

```
LINKAGE SECTION.
01  IO-PCB.
    05  IO-PCB-LTERM          PIC X(8).
    05  FILLER                PIC X(2).
    05  IO-PCB-STATUS        PIC X(2).
    05  IO-PCB-INPUT-PREF.
        10  IO-PCB-DATE      PIC S9(7) COMP-3.
        10  IO-PCB-TIME      PIC S9(7) COMP-3.
        10  IO-PCB-MSG-SEQ   PIC S9(7) COMP.
    05  IO-PCB-MOD-NAME      PIC X(8).
    05  IO-PCB-USER-ID      PIC X(8).
```

In an APS IMS/SQL program, code a PSB with the same name as your DB2 subschema. The PSB must specify an IO PCB (by setting CMPAT=YES in the PSBGEN). If your program uses MSG-SW, the PSB must also specify an alternate IO PCB.

- Comments:**
- A program requires IO PCBs to obtain an input message and return a reply to the originating terminal.
 - To send a message to a logical terminal instead of to the originating terminal, code the MSG-SW call and specify an alternate IO PCB that you define with MODIFY=YES in the program specification block (PSB). MSG-SW cannot include the EXPRESS keyword when it references this alternate IO PCB.

Program DB/DC Report (PG02)

Category: APS-generated report (see *Application Reports*)

Description: The Program DB/DC Report includes documentation summaries on the subschemas, PSBs, and screens used by a program. The Program DB/DC Report has a separate section for database views, record I/O areas, and screen I/O areas.

The Database Views section provides the following information.

- Record names of the root and each dependent segment
- Segments in hierarchical sequence, indented to show each segment's level in the database
- Functions (PROCOPTS) that can be performed for each segment
- Key and sequence fields for accessing each record
- VSAM file descriptions

The Record I/O Areas section provides information on the COBOL I/O areas for the database and file records. Records are listed alphabetically by record name.

The Screen I/O Areas section provides information on COBOL I/O areas for up to 20 screens arranged alphabetically.

Comment: Produce the Program DB/DC Report from the Documentation Facility.

Example:

```
REPORT CODE: PG02          APS PROGRAMMER SUBSYSTEM          01/17/92          14:29
                          PROGRAM DB/DC REPORT
```

```
*****
PROGRAM: TDOM  PSB/SUBSCHEMA: TDDB2
SCREENS: TDOM
*****
```

```
TABLES/VIEWS: (DB2 RECORDS)
```

```
-----
```

```
TABLE: TDCUST-REC
TABLE: TDODET-REC
```

Reference

TABLE: TDORDR-REC
 TABLE: TDPART-REC

RECORD IO AREAS

RECORD: TDCUST-REC

```
*****
*          DCLGEN TABLE   : MKTAEA.TDCUST USER: CLSTR1      *
*          LIBRARY: CLSAPS.CLS2.COPYLIB                      *
*          MEMBER  : TDCUST  DATE: 90/11/27                  *
*          ACTION(REPLACE) TIME 13:11:44                    *
*          DB2 SYSTEM: DB2B                                  *
*****
```

```
EXEC SQL DECLARE MKTAEA.TDCUST TABLE
(CM_CUSTOMER_NO CHAR(6) NOT NULL,
 CM_CUSTOMER_NAME CHAR(20) ,
 CM_CUSTOMER_ADDR CHAR(20) ,
 CM_CUSTOMER_CITY CHAR(20) ,
 CM_CUSTOMER_ZIP CHAR(9) )
END-EXEC.
```

```
*****
* COBOL DECLARATION FOR TABLE MKTAEA.TDCUST                *
*****
```

```
01 TDCUST-REC.
 10 CM-CUSTOMER-NO PIC X(6).
 10 CM-CUSTOMER-NAME PIC X(20).
 10 CM-CUSTOMER-ADDR PIC X(20).
 10 CM-CUSTOMER-CITY PIC X(20).
 10 CM-CUSTOMER-ZIP PIC X(9).
```

```
*****
* THE NUMBER OF COLUMNS DESCRIBED BY THE DECLARATION IS 5      *
*****
```

SCREEN IO AREAS

```
SCREEN: TDOM                                FORMAT (:/D/M )
                                           RULEDLINE:
                                           INITIAL CURSOR (X):: :
                                           COLOR: :: :
UNDERSCORE (U/::):: : : :
BLINKING (B/::):: : : : :
```

```

REVERSE VIDEO (R/): : : : : : :
INTENSITY (B/:/D): : : : : : :
PROTECT (P/U): : : : : : :
01 TDOM-RECORD. : : : : : : :
05 TDOM-FUNCTION PIC X(1). U B : : : : X : :
05 TDOM-ORDER-NO PIC X(6). U B : : : : : : :
05 TDOM-SAVEKEY-1 PIC X(8). P D : : : : : : :
05 TDOM-CUSTOMER-NO PIC X(6). U B : : : : : : :
05 TDOM-CUSTOMER-NAME PIC X(20). U B : : : : : : :
05 TDOM-CUST-ENTRY-DATE PIC X(8). U B : : : : : : :
05 TDOM-ORDER-DEL-DUE-D PIC X(8). U B : : : : : : :
05 TDOM-ORDER-DEL-INSTR PIC X(20). U B : : : : : : :
05 TDOM-TABLE-1. : : : : : : :
    10 FILLER OCCURS 5. : : : : : : :
    15 TDOM-ROW-FUNCTION
        PIC X(1).
    15 TDOM-PART-NO PIC X(8).
    15 TDOM-LINE-NO PIC X(4).
    15 TDOM-PART-SHORT-DESC
        PIC X(14).
    15 TDOM-QTY-ORDERED
        PIC X(8).
    15 TDOM-QTY-BASE-PRICE
        PIC X(10).
    15 TDOM-TAX-CATEGORY
        PIC X(1).
    15 TDOM-SAVEKEY-2 PIC X(15).
    05 TDOM-MESSAGE PIC X(79). P B : : : : : : :

```

Program Definition Report (PG01)

Category: APS-generated report (see *Application Reports*)

Description: The Program Definition Report produces a listing of the program code you create in the Program Painter. Use this report when you need to review program listings to troubleshoot problems, or when you need to document completed programs in your applications.

Comment: Produce the Program Definition Report from the Report Generator, Painter Menu, or Application Painter.


```
XCTL programname [errorpara]
... [DLIUIB pcbname [pcbname ...]]
```

The invoking program must pass the PCBs in the order that they are coded in the Linkage Section of the invoked program.

When a program invokes a LINK to a subprogram, and passes a scheduled PSB, it expects the PSB to remain scheduled when control returns from the subprogram. To return without terminating the PSB, use TERM because it does not terminate a PSB passed from a higher-level program. To terminate a PSB, use the CIC-TERM-PSB call.

Example: Pass part of a scheduled PSB and two Linkage Section data areas.

```
LINK PROGRMB * DLIUIB
... ORDERDB-PCB
... ITEMDB-PCB
... USER-LINK-1
... USER-LINK-2
```

The PSB remains scheduled at the start of the next program.

Project and Group Options

Category: Application generation

Description: Identify application project and group location and where you want APS to generate the project and group DDIFILE dataset.

Procedure:

- 1 Access the Project Group Environment screen. To do so, from the APS Options Menu, enter option **2** in the **Option** field. Alternatively, from any APS screen enter **opt 2** in the **Command** field. The Project Group Environment screen displays.

- 2 Complete the fields on the Project Group Environment screen.

Field	Description
Project	The name of the project. For example, MYPROJ. Must be 1-8 alphanumeric characters; the first character must be alphabetic.

Field	Description
Group	The name of the group. For example, mygrp. Must be 1-8 alphanumeric characters; the first character must be alphabetic.
DDIFILE	The location of the project and group DDIFILE data set; do not specify the name DDIFILE. Default: The project and group path specified above. For example, myproj.group.
Data Element Library Prefix	Optional. The location of the Data Element Facility APSDE data set; do not specify the name APSDE. For example, APSPG.PROJECT1.GROUP1.

REC

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Description:

Define a data -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
REC *datastructure*

Parameter: *datastructure* Valid Data Structure Painter construct.

- Comments:**
- Each REC keyword generates an 01-level data structure. APS transfers the REC entity to the Working-Storage Section.
 - The preceding section keyword determines the placement of a data structure in the generated program. Associated section keywords are

FD	File Section (see <i>FD</i>)
SD	Sort File Description (see <i>SD</i>)
WS	Working-Storage Section (see <i>WS</i>)
LK	Linkage Section (see <i>LK</i>)

- Code a COBOL COPY with REC, as follows.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
REC    dataname
        COPY copybookname
        ... REPLACING field1 BY field2
```

RED

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Compatibility: Batch environments

Description: Name and begin the statement block that defines the report; identify the report name, control fields, and report page characteristics.

Syntax:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
RED    reportname
        [CODE literal]
        [CONTROL [IS] [FINAL] dataname /
         CONTROLS [ARE] [FINAL] dataname1 ... datanameN ]
        [WRITE ROUTINE [IS] paragraphname]
        [PAGE LIMIT[S] IS|ARE number LINE[S]
         [FIRST DETAIL linenumber]
         [LAST DETAIL linenumber]
         [FOOTING linenumber]] [.]
```

Parameter: *reportname* Report name of the REPORT clause in the File Section

- Comments:**
- Each report requires RED statement. Reportnames in both the RED and FD statements must be identical.
 - Code RED clauses in any sequence.
 - If reportname exceeds 20 characters, Report Writer creates an abbreviated record name, consisting of:
 - The first character of each hyphenated word of the report name, except the last word
 - A hyphen

- The last word
- RECORD

For example:

Input REALLY-LONG-REPORT-NAME

Output RLR-NAME-RECORD

Example:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
RED    YEAR-END-SALES-SUMMARY
        CONTROLS ARE FINAL
        REGION REGION-MGR-FLD OFFICE
        PAGE LIMIT IS 58 LINES
        FIRST DETAIL 9
        HEADING 1
        FOOTING 58.
```

REDEFINES

Category: Data Structure Painter construct (see *Data Structures*)

Description: Code clauses in your data structures to redefine elements.

Syntax: *dataname* REDEF[INES] |R *PICformat*

Parameter: *PICformat* PICTURE format for data name being redefined

Comment: When identifying controls in the CONTROL clause, *dataname* must be an elementary data name. In the following example, B cannot be used as a control variable because it is a group data item. To make B into an elementary data item, use the REDEFINES clause.

```
WS01  A                               PIC X(2).
WS01  B.
        02  B-1                       PIC 9(4).
        02  B-2                       PIC 9(4).
WS01  B-REDEF  REDEFINES B           PIC X(8).
.
.
.
```

```
RED TEST-REPORT
CONTROLS ARE A B-REDEF
```

Example: Data Structure Painter format:

```
-LINE- ----- Data Structure Painter -----
000001          WRK1-FIELD-8
000002          WRK1-FIELD-9      X4
000003          WRK1-FIELD-10    X(30)
000004          WRK1-FIELD-11    R X34
```

Generated COBOL code:

```
01  WRK1-FIELD-8.
    05  WRK1-FIELD-9          PIC X(4) .
    05  WRK1-FIELD-10        PIC X(30) .
01  WRK1-FIELD-11  REDEFINES WRK1-FIELD-8
                        PIC X(34) .
```

REFERENCE

Category: Report Writer clause (see *Report Writer Structures* and the *APS User's Guide* chapter *Creating Reports with Report Writer*.)

Compatibility: Batch environments

Description: Establish summing capability for a non-printing detail item and sum the item in a control footing.

Syntax: R[EFERENCE] [IS] *dataname* PIC[TURE] [IS] *picclause*
 [DATA-NAME [IS] *fieldname*]

Parameters: *dataname* Data item being referenced

DATA-NAME *fieldname* Name a sum accumulator established by a SUM or REFERENCE clause. Do not define *fieldname* in Working-Storage. At generation, APS inserts *fieldname* after the level number in the report group. DATA-NAME moves the value of the internal SUM accumulator to *fieldname*.

Code DATA-NAME when a SUM UPON clause references a DETAIL report group, when the program references a sum accumulator, or when a sum accumulator requires a data name for qualification.

PIC *picclause*

Specify the format of dataname. If dataname is a report mock-up field instead of a Working-Storage field, the next matching COBOL picture in the report mock-up is the picclause for dataname.

- Comments:**
- When you code a REFERENCE statement, the PIC clause must match the PIC clause in the record description.

For example:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----
01     COST-DETAIL TYPE DETAIL
      MOCKUP LINE 9
      SOURCE WS-DEPT
      SOURCE WS-EMPLOYEE
      SOURCE WS-CITY
      REFERENCE EMP-CTR    PIC 999
01     TYPE CONTROL FOOTING
      MOCKUP LINE 9
      SOURCE WS-DEPT
      SUM EMP-CTR
WS01  EMP-CTR          PIC 999    VALUE 1.
```

If one of the PIC clauses were PIC 9(3), Report Writer would not find a match.

- In a REFERENCE statement, the data item referenced must be defined in Working-Storage with a VALUE clause. The value in the VALUE clause tells Report Writer the increment to add to the internal accumulator each time the detail line prints. In the previous example, APS adds 1 to the internal accumulator whenever the detail line prints.
- The generated program does not describe the referenced field, dataname, on the report mock-up detail line, nor display it on the printed detail line.

Example:

LOCATION	LAST COUNT DATE	QUANTITY IN STOCK	QUANTITY ISSUED	QUANTITY RECEIVED
XXXXXXXXXXXXXX	99/99/99	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9
TOTAL BY LOCATION:		Z,ZZZ,ZZ9	Z,ZZZ,ZZ9	Z,ZZZ,ZZ9
TOTAL NUMBER OF SALES BY LOCATION: ZZZ,ZZ9				
-KYWD- 12-*----20---*----30---*----40---*----50---*----60				
01	DETAIL-LINE TYPE IS DETAIL.			
	MOCKUP LINE 16			
	SOURCE WS-LOCATION-CODE	GROUP INDICATE		
	SOURCE WS-LAST-COUNT-MONTH	PIC 99		
	SOURCE WS-LAST-COUNT-DAY	PIC 99		
	SOURCE WS-LAST-COUNT-YEAR	PIC 99		
	SOURCE WS-QTY-IN-STOCK			
	SOURCE WS-QTY-ISSUED			
	SOURCE WS-QTY-RECEIVED			
	REFERENCE WS-NO-OF-SALES	PIC 9999		
01	TYPE IS CONTROL FOOTING WS-LOCATION-CODE			
	MOCKUP LINES 17 THRU 20			
	SUM WS-NO-OF-SALES	PIC ZZZ9		

REM**Category:** Program Painter and Specification Editor keyword (see *Keywords*)**Compatibility:** Supported for COBOL only, not COBOL/2**Description:** Create Identification Division Comments: text.

Syntax:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
REM      commentline1
      .
      .
      .
      commentlineN
```

REPEAT

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Establish a loop for testing; use with WHILE or UNTIL statement to test the loop at the middle or at the end. This construction eliminates the need for GO TO statements and the multiple tests that are required to form similar loops in COBOL.

Syntax: Format 1:

```
REPEAT
    statementblock
UNTIL|WHILE condition
    [ statementblock ]
```

Format 2:

```
REPEAT VARYING|LINKING indexname|identifier1
... [FROM indexexpression]|[arithmeticexpression]
... [BY literal]|[identifier2]
    statementblock
UNTIL|WHILE condition
    [ statementblock ]
```

Format 3:

```
REPEAT VARYING indexname|identifier1
... [FROM indexexpression]|[arithmeticexpression1]
... [BY literal]|[identifier2]
... [DOWN] TO|THRU arithmeticexpression2
    [ statementblock ]
```

Format 4:

```
REPEAT LINKING indexname|identifier1
... [FROM indexexpression]|[arithmeticexpression1]
... BY identifier2
... [DOWN] TO arithmeticexpression2
    [ statementblock ]
```

Format 5:

```

REPEAT VARYING|LINKING clause1
      . . .
[... VARYING|LINKING clauseN]
[ statementblock ]
UNTIL|WHILE condition
[ statementblock ]

```

Parameters:	<i>arithmeticexpression</i>	A legal arithmetic relation-condition
	<i>indexexpression</i>	Format can be: <i>literal</i> <i>Identifier +/- literal</i> <i>indexname</i>
	<i>identifier2</i>	Names a table entry, such as a data name with an OCCURS clause

Logic Execution:

- The statement block subordinate to the REPEAT, and any statement block subordinate to the WHILE or UNTIL, forms the loop and executes under control of the conditions specified with WHILE or UNTIL.
- If the WHILE or UNTIL does not have a subordinate statement block, the condition is tested at the end of the loop. Control returns to the beginning of the REPEAT statement block until the WHILE condition is false or the UNTIL condition is true. The next statement executed is the first one with the same or less indentation than the REPEAT/WHILE or REPEAT/UNTIL.
- If the WHILE or UNTIL has a subordinate statement block, it establishes a loop which contains a test in the middle. When the WHILE condition is FALSE or the UNTIL condition is true, control passes out of the loop to the next statement with the same or less indentation than the REPEAT, without executing its statement block.
- Coding an extraneous WHILE or UNTIL causes an endless loop, if the looping condition is satisfied by the REPEAT VARYING/LINKING preceding it.

- In the following format, APS tests the condition after the REPEAT *statementblock1* executes.

```
REPEAT
    statementblock
UNTIL condition
    statementblock3
```

Be careful using this format for reading records—it can read the last record twice.

- In the following format, APS tests the condition after the REPEAT *statementblock1* executes, but before the UNTIL *statementblock2* executes. When the UNTIL condition is true, the UNTIL *statementblock2* does not execute.

```
REPEAT
    statementblock
UNTIL condition
    statementblock

    statementblock3
```

- DOWN is generally used if *literal* or *identifier2* is negative at execution, and used if positive. The loop executes until UNTIL *indexname identifier1 < arithmeticexpression2*
- With the TO option, the loop executes to, but not including, the stop-point. With the THRU option, the loop executes through and including the stop-point.

Comments:

- If FROM is not coded, the default is the value of *indexname* or *identifier1* at the time of execution.
- If BY is not coded, the default is BY 1 (or -1, if DOWN TO/THRU is coded).
- APS initializes the index or identifier is immediately before the REPEAT loop begins and increments it each time the loop repeats, immediately before the statement block repeats.
- For the identifier to be treated as an index, the indexed data element structure must be present in the program during APS precompilation, otherwise subscript processing is assumed.
- To copy data containing an indexed structure, use the % INCLUDE statement.

Examples: Read header records from a file, move the relevant data to a table, and then print the table.

```

-LINE- -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----
002010  NTRY
002020      LINE-SUB = 1
002030      PRINT-TABLE = SPACES
002040      OPEN INPUT INPUT-FILE
002050      ... OUTPUT PRINT-FILE
002070      /* BEGIN FIRST LOOP
002080      REPEAT
002090          READ INPUT-FILE
002100      UNTIL AT END ON INPUT-FILE
002110          IF REC-TYPE = 'HDR'
002120              WORK-FIELD = INPUT-DATA
002130              IF WORK-FIELD NOT = SPACES
002140                  COLUMN-SUB = 0
002150                  /* BEGIN SECOND LOOP
002160                  REPEAT
002170                      COLUMN-SUB = COLUMN-SUB + 1
002180                      PRINT-COL (LINE-SUB, COLUMN-SUB) =
002190                      ... WORK-FIELD-CHAR (COLUMN-SUB)
002200                      UNTIL COLUMN-SUB = COLUMN-SUB-LIMIT
002210                      ... OR WORK-FIELD-CHAR (COLUMN-SUB) =
002211                      ... '/'
002220                      /* END OF SECOND LOOP
002230                      LINE-SUB = LINE-SUB + 1
002240      /* END OF FIRST LOOP
002250
002260      PERFORM WRITE-PRINT-TABLE
002270      CLOSE INPUT-FILE PRINT-FILE
002280
002290  PARA  WRITE-PRINT-TABLE
002300          LINE-SUB = 1
002310          WHILE PRINT-LINE (LINE-SUB) NOT = SPACES
002320              ... AND LINE-SUB NOT > LINE-SUB-LIMIT
002330              WRITE PRINT-REC FROM PRINT-LINE (LINE-SUB)
002340              LINE-SUB = LINE-SUB + 1

```

Perform the same function as the second loop in the preceding example, but use **VARYING** to set and increment **COLUMN-SUB**.

```

12-*-----20---*-----30---*-----40---*-----50---*-----
      IF WORK-FIELD NOT = SPACES
          REPEAT VARYING COLUMN-SUB FROM 1 BY 1
              PRINT-COL (LINE-SUB, COLUMN-SUB) =
              ... WORK-FIELD-CHAR (COLUMN-SUB)

```

```

UNTIL COLUMN-SUB = COLUMN-SUB-LIMIT
... OR WORK-FIELD-CHAR (COLUMN-SUB) = '/'
LINE-SUB = LINE-SUB + 1

```

Use REPEAT ... VARYING to move data items in diagonal sequence (upper right to lower left) from a two-dimensional table to a one-dimensional table. Terminate the loop when after DOWN THRU.

```

12-*----20---*----30---*----40---*----50---*----
  REPEAT VARYING ROW-SUB FROM 1 BY 1
  ... VARYING COLUMN-SUB FROM 5 DOWN THRU 1
  X-FIELD (ROW-SUB) =
  ... TABLE-ELEMENT (ROW-SUB, COLUMN-SUB)

```

Use II as the pointer, MY-CHAIN for the initial setting, BLOCK-LINK for the linking element in the table, and ZERO to establish when to stop.

```

-KYWD- 12-*----20---*----30---*----40---*----50---*----60
.
01  BLOCK-STRUCTURE.
    02  BLOCK OCCURS 250 TIMES.
        03  BLOCK-LINK PIC S9(9) COMP SYNC.
        03  BLOCK-DATA PIC X(20).
.
NTRY
    REPEAT LINKING II FROM MY-CHAIN
    ... BY BLOCK-LINK TO ZERO
    PERFORM PASS-DATA( BLOCK-DATA (II))
.
PARA  PASS-DATA(+PASS-DATA-BLOCK)
.

```

Generated COBOL code:

```

    MOVE MY-CHAIN TO II.
    GO TO G--002.
G--001.
    MOVE BLOCK-LINK (II) TO II.
G--002.
    IF II NOT = ZERO
        MOVE BLOCK-DATA (II) TO PASS-DATA-BLOCK
        PERFORM PASS-DATA THRU PASS-DATA--XIT
        GO TO G--001.

```

APS uses SET in the generated code if the subscript is an index; otherwise, APS uses MOVE as shown above.

Report Mock-Ups

Category: Report layout associated with Report Writer program

Description: Paint report layouts, called mock-ups, in the Report Painter. Define the mock-up by typing literals and output fields to visually represent the report output. Specify the following COBOL and COBOL/2 output edit masks directly within the report mock-up:

- Floating numeric formats
- Alphanumeric formats

- Mock-Up Rules:**
- Paint the mock-up in columns 1 to 247.
 - Paint lines as they appear within their report group.
 - Include any blank lines that appear between the first and last lines of a report group in the mock-up.
 - To minimize programming, include blank lines preceding or following a report group in the mock-up.
 - Paint the line or lines that compose a DETAIL group only once.
 - Each field that composes a line must be shown with a COBOL picture for variable data or with the literal for fixed data.
 - Within the layout of a line, position each field in the desired printing position with exact spacing between fields.
 - All COBOL picture characters are available for output fields except A, which is considered a literal character anywhere in the mock-up.
 - APS considers a string of hyphens a literal because of its frequent use for underlining.
 - APS considers any single COBOL picture character, that is preceded and followed by a space, a literal. Exception to this rule: 9 and X.
 - APS considers a single COBOL picture character, such as -, X, Z, or 9, that is embedded in a string of non-blank, non-picture characters as part of a literal. For example, the following are literals:

```
1979  
WXYZ
```

```
EXTRA
WIZARD
```

and the following are pictures beside literals:

```
#99      Literal is #, PIC is 99.
1999     Literal is 1, PIC is 999.
Section-999  Literal is SECTION, PIC is -999.
```

- Show a floating-point item in the mock-up with a COBOL picture in floating-point form. The same picture must be repeated in a PIC phrase in the corresponding SOURCE, SUM, or VALUE statement.
- Use PIC clauses instead of COBOL masks when formatting dates and times containing / \$ or :.

Example:

```
EDIT --- REPORT: REPORT ----- COLUMNS 001 @
COMMAND ==> SCROLL ==> CSA
***** TOP OF DATA *****
000100 WONDERFUL WIDGETS INCORPORATED
000200
000300 STOCK REPORT
000400
000500 XXXXXX XXX
000600
000700 MID ATLANTIC
000800 STOCK REPORT
000900 XXXXXX XXX
001000
001100 LOCATION          LAST COUNT   QUANTITY IN   QUANTITY
001200                   DATE          STOCK          ISSUED        RECEIVED
001300
001400 XXXXXXXXXXXXX      99/99/99      777,779       777,779       777,779
001500
001600
001700 TOTAL BY LOCATION:          2,222,229     2,222,229     2,222,229
001800
001900 TOTAL NUMBER OF SALES BY LOCATION: 777,779
***** BOTTOM OF DATA *****
```

Report Sample Program and Mock-Up

Category: Report Writer program and Report Painter mock-up

Description: Illustrate Report Writer structures and the use of iterative expressions.

Report Mock-Up:

```

=====
                                EGAS, INC.

                                XXXX YEAR-END PRODUCT SALES SUMMARY

*****

                                PAGEZZZZ9
                                REGION:  XXXXXXXXX
                                XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

PRODUCT      ----- 1-ST QUARTER ----- 2-ND QUARTER ----- 3-RD QUARTER ----- 4-TH QUARTER -----
              JAN  FEB  MAR  TOTAL  APR  MAY  JUN  TOTAL  JUL  AUG  SEP  TOTAL  OCT  NOV  DEC  TOTAL  TOTAL
-----

SALES OFFICE: XXXXXXXXXXXXXXXXXXXX
MANAGER:      XXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXX  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  $$$$,$$$

TOTAL FOR:
XXXXXXXXXXXXXXXXXXXX  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  $$$$,$$$

TOTAL FOR:
XXXXXXXXXX          Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  $$$$,$$$

TOTAL              Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  Z,ZZ9 Z,ZZ9 Z,ZZ9 ZZ,ZZ9  $$$$,$$$

                                EGAS! BUY FROM EGAS!

*****

                                EGAS! ANOTHERBANNER YEAR!

=====

```

Program Painter Source:

```

REM  READS DATA.EXTRACT AND GENERATES A SUMMARY
      REPORT BY REGION, OFFICE AND PRODUCT.

IO   EXTRACT-FILE                ASSIGN TO UT-S-EXTRACT
IO   SALES-SUMMARY-FILE          ASSIGN TO UT-S-SUMMREPT
IO   DEFINITION-FILE            ASSIGN TO UT-S-DEFS
FD   EXTRACT-FILE

      LABEL RECORDS ARE STANDARD
      RECORDING MODE IS F
      RECORD CONTAINS 90 CHARACTERS
      BLOCK CONTAINS 0 RECORDS.

01   EXTRACT-FILE-RECORD
      02  EXT-REGION                PIC X(9).
      02  EXT-OFFICE                 PIC X(15).
      02  EXT-PRODUCT               PIC X(18).
      02  EXT-SALES-DOLLARS         PIC 9(4) OCCURS 12.

FD   SALES-SUMMARY-FILE
      LABEL RECORDS ARE STANDARD
      RECORDING MODE IS F

```

RECORD CONTAINS 133 CHARACTERS
 BLOCK CONTAINS 0 RECORDS
 REPORT IS YEAR-END-SALES-SUMMARY.

```

FD  DEFINITION-FILE
    LABEL RECORDS ARE STANDARD
    RECORDING MODE IS F
    RECORD CONTAINS 38 CHARACTERS
    BLOCK CONTAINS 0 RECORDS.

01  DEFINITION-RECORD.
    02  DEFINITION-TYPE          PIC X(3).
    02  DEFINITION-REGION       PIC X(29).
    02  FILLER                  PIC X(6).

01  DEFINITION-RECORD-2.
    02  FILLER                  PIC X(3).
    02  DEFINITION-OFFICE       PIC X(35).

WS01 JJ                        PIC S9(4) COMP SYNC VALUE ZERO.
WS01 II                        PIC S9(4) COMP SYNC VALUE ZERO.
WS01 FIRST-FLG                PIC X(1) VALUE 'T'.

WS01 REGION-DEFINITIONS.
    02  REGION-TABLE OCCURS 4 TIMES INDEXED BY REGION-IDX.
        03  REGION-NAME          PIC X(9).
        03  REGION-MANAGER       PIC X(20).

WS01 REGION                    PIC X(9).
WS01 REGION-MGR.
    02  REG-MGR OCCURS 20 TIMES
        INDEXED BY REG-MGR-IDX
        PIC X(1).

WS01 REG-MGR-MAX                PIC S9(4) COMP SYNC VALUE +20.
WS01 REGION-MGR-FIELD.
    02  REGION-MGR-X OCCURS 30 TIMES INDEXED BY MGR-IDX
        PIC X(1).

WS01 REGION-MGR-FLD REDEFINES REGION-MGR-FIELD
        PIC X(30).

WS01 REGION-MGR-MAX            PIC S9(4) COMP SYNC VALUE +30.
WS01 MANAGER-WORD              PIC X(10) VALUE 'MANAGER: '.
WS01 MANAGER-BY-CHAR REDEFINES MANAGER-WORD.
    02  MANAGER-LETTER OCCURS 10 TIMES
        INDEXED BY LETTER-IDX
        PIC X(1).

WS01 MANAGER-WORD-SIZE         PIC S9(4) COMP SYNC VALUE +10.
  
```

WS01 OFFICE-DEFINITIONS.
 02 OFFICE-TABLE OCCURS 14 TIMES INDEXED BY OFFICE-IDX.
 03 OFFICE-NAME PIC X(15).
 03 OFFICE-MANAGER PIC X(20).

WS01 OFFICE PIC X(15).
 WS01 OFFICE-MGR PIC X(20).

WS01 QTR-1-SALES-DOLLARS PIC 9(5) VALUE ZERO.
 WS01 QTR-2-SALES-DOLLARS PIC 9(5) VALUE ZERO.
 WS01 QTR-3-SALES-DOLLARS PIC 9(5) VALUE ZERO.
 WS01 QTR-4-SALES-DOLLARS PIC 9(5) VALUE ZERO.

WS01 YR-SALES-DOLLARS PIC 9(6) VALUE ZERO.
 WS01 CURRENT-DATE-X.
 02 CURRENT-YEAR PIC 9(2).
 02 FILLER PIC X(4).

WS01 REPORT-YEAR.
 02 FILLER PIC 9(2) VALUE 19.
 02 REPORT-YEAR-X PIC 9(2).

RED YEAR-END-SALES-SUMMARY
 CONTROLS ARE FINAL REGION REGION-MGR-FLD OFFICE
 PAGE LIMIT IS 58 LINES
 FIRST DETAIL 9
 HEADING 1
 FOOTING 58.

MOCK SUMMARY

01 RH-YEAR-END-SALES-SUMMARY TYPE IS REPORT HEADING
 NEXT GROUP IS NEXT PAGE.
 MOCKUP LINES 1 THRU 4
 LINE 25
 SOURCE REPORT-YEAR PIC X(4).

01 PH-YEAR-END-SALES-SUMMARY TYPE IS PAGE HEADING.
 MOCKUP LINES 10 THRU 17
 SOURCE PAGE-COUNTER PIC ZZZZ9.
 SOURCE REGION PIC X(9).
 SOURCE REGION-MGR-FLD PIC X(30).

01 CH-REGION TYPE IS CONTROL HEADING
 REGION
 NEXT GROUP IS NEXT PAGE.

01 CH-OFFICE TYPE IS CONTROL HEADING
OFFICE.
MOCKUP LINES 18 THRU 20
SOURCE OFFICE PIC X(15).
SOURCE OFFICE-MGR PIC X(20).

01 DE-YEAR-END-SALES-SUMMARY TYPE IS DETAIL.
MOCKUP LINE 21
SOURCE EXT-PRODUCT PIC X(8).
SOURCE EXT-SALES-DOLLARS (#1/3) PIC Z,ZZ9
SOURCE QTR-1-SALES-DOLLARS PIC ZZ,ZZ9
SOURCE EXT-SALES-DOLLARS (#4/6) PIC Z,ZZ9
SOURCE QTR-2-SALES-DOLLARS PIC ZZ,ZZ9
SOURCE EXT-SALES-DOLLARS (#7/9) PIC Z,ZZ9
SOURCE QTR-3-SALES-DOLLARS PIC ZZ,ZZ9
SOURCE EXT-SALES-DOLLARS (#10/12) PIC Z,ZZ9
SOURCE QTR-4-SALES-DOLLARS PIC ZZ,ZZ9
SOURCE YR-SALES-DOLLARS PIC \$\$\$\$,\$\$\$

01 PF-YEAR-END-SALES-SUMMARY TYPE IS PAGE FOOTING
NEXT GROUP IS NEXT PAGE.
MOCKUP LINE 32

01 RF-YEAR-END-SALES-SUMMARY TYPE IS REPORT FOOTING.
MOCKUP LINE 38
LINE IS 25

01 CF-FINAL TYPE IS CONTROL FOOTING
FINAL.
MOCKUP LINES 29 THRU 31
SUM EXT-SALES-DOLLARS (#1/3) PIC Z,ZZ9
SUM QTR-1-SALES-DOLLARS PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#4/6) PIC Z,ZZ9
SUM QTR-2-SALES-DOLLARS PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#7/9) PIC Z,ZZ9
SUM QTR-3-SALES-DOLLARS PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#10/12) PIC Z,ZZ9
SUM QTR-4-SALES-DOLLARS PIC ZZ,ZZ9
SUM YR-SALES-DOLLARS PIC \$\$\$\$,\$\$\$

01 CF-REGION TYPE IS CONTROL FOOTING
REGION.
MOCKUP LINES 26 THRU 28
SOURCE REGION PIC X(9)
SUM EXT-SALES-DOLLARS (#1/3) PIC Z,ZZ9
SUM QTR-1-SALES-DOLLARS PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#4/6) PIC Z,ZZ9
SUM QTR-2-SALES-DOLLARS PIC ZZ,ZZ9

```

SUM EXT-SALES-DOLLARS (#7/9)    PIC Z,ZZ9
SUM QTR-3-SALES-DOLLARS        PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#10/12) PIC Z,ZZ9
SUM QTR-4-SALES-DOLLARS        PIC ZZ,ZZ9
SUM YR-SALES-DOLLARS           PIC $$$$,$$$

01  CF-OFFICE                    TYPE IS CONTROL FOOTING
                                   OFFICE.

MOCKUP LINES 23 THRU 25
SOURCE OFFICE                    PIC X(15)
SUM EXT-SALES-DOLLARS (#1/3)    PIC Z,ZZ9
SUM QTR-1-SALES-DOLLARS        PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#4/6)    PIC Z,ZZ9
SUM QTR-2-SALES-DOLLARS        PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#7/9)    PIC Z,ZZ9
SUM QTR-3-SALES-DOLLARS        PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#10/12) PIC Z,ZZ9
SUM QTR-4-SALES-DOLLARS        PIC ZZ,ZZ9
SUM YR-SALES-DOLLARS           PIC $$$$,$$$

DPAR SUPPRESS CH-REGION SECTION
USE BEFORE REPORTING CH-REGION
DPAR SUPPRESS CH-REGION-PARA
IF FIRST-FLG = TRUE
    SUPPRESS PRINTING

PROC
ACCEPT CURRENT-DATE-X FROM DATE
MOVE CURRENT-YEAR TO REPORT-YEAR-X

PERFORM LOAD-DEFINITIONS

OPEN INPUT EXTRACT-FILE
OPEN OUTPUT SALES-SUMMARY-FILE

INITIATE YEAR-END-SALES-SUMMARY
MOVE ZERO TO PAGE-COUNTER

REPEAT
    READ EXTRACT-FILE
UNTIL AT END ON EXTRACT-FILE
    IF EXT-OFFICE NOT = OFFICE
        PERFORM LOCATE-MANAGERS

ADD EXT-SALES-DOLLARS (1) TO QTR-1-SALES-DOLLARS
ADD EXT-SALES-DOLLARS (2) TO QTR-1-SALES-DOLLARS
ADD EXT-SALES-DOLLARS (3) TO QTR-1-SALES-DOLLARS

```

```
ADD EXT-SALES-DOLLARS (4) TO QTR-2-SALES-DOLLARS
ADD EXT-SALES-DOLLARS (5) TO QTR-2-SALES-DOLLARS
ADD EXT-SALES-DOLLARS (6) TO QTR-2-SALES-DOLLARS
```

```
ADD EXT-SALES-DOLLARS (7) TO QTR-3-SALES-DOLLARS
ADD EXT-SALES-DOLLARS (8) TO QTR-3-SALES-DOLLARS
ADD EXT-SALES-DOLLARS (9) TO QTR-3-SALES-DOLLARS
```

```
ADD EXT-SALES-DOLLARS (10) TO QTR-4-SALES-DOLLARS
ADD EXT-SALES-DOLLARS (11) TO QTR-4-SALES-DOLLARS
ADD EXT-SALES-DOLLARS (12) TO QTR-4-SALES-DOLLARS
```

```
REPEAT VARYING II FROM 1 BY 1
UNTIL II > 12
    ADD EXT-SALES-DOLLARS (II) TO YR-SALES-DOLLARS
```

```
GENERATE DE-YEAR-END-SALES-SUMMARY
```

```
MOVE FALSE TO FIRST-FLG
```

```
MOVE ZEROES TO QTR-1-SALES-DOLLARS
... QTR-2-SALES-DOLLARS QTR-3-SALES-DOLLARS
... QTR-4-SALES-DOLLARS YR-SALES-DOLLARS
```

```
TERMINATE YEAR-END-SALES-SUMMARY
```

```
CLOSE EXTRACT-FILE SALES-SUMMARY-FILE
```

```
PARA LOAD-DEFINITIONS.
```

```
SET REGION-IDX TO 1
SET REGION-IDX DOWN BY 1
SET OFFICE-IDX TO 1
SET OFFICE-IDX DOWN BY 1
```

```
OPEN INPUT DEFINITION-FILE
```

```
REPEAT
    READ DEFINITION-FILE
UNTIL AT END ON DEFINITION-FILE
```

```
IF DEFINITION-TYPE = 'REG'
    SET REGION-IDX UP BY 1
    MOVE DEFINITION-REGION
        TO REGION-TABLE (REGION-IDX)
```

```
ELSE-IF DEFINITION-TYPE = 'OFF'
```

```

                SET OFFICE-IDX UP BY 1
                MOVE DEFINITION-OFFICE
                                TO OFFICE-TABLE (OFFICE-IDX)

CLOSE DEFINITION-FILE

PARA LOCATE-MANAGERS.
SET REGION-IDX TO 1
SEARCH REGION-TABLE
WHEN EXT-REGION = REGION-NAME (REGION-IDX)
    MOVE REGION-MANAGER (REGION-IDX) TO REGION-MGR
MOVE SPACES TO REGION-MGR-FLD

SET REG-MGR-IDX TO REG-MGR-MAX
WHILE REG-MGR (REG-MGR-IDX) = SPACE
... AND REG-MGR-IDX > ZERO
    SET REG-MGR-IDX DOWN BY 1
SET JJ TO REG-MGR-IDX

COMPUTE II =
... (REGION-MGR-MAX - MANAGER-WORD-SIZE - JJ) / 2
ADD 1 TO II

IF II <= ZERO
    MOVE 1 TO II
SET REG-MGR-IDX TO 1
SET LETTER-IDX TO 1

REPEAT VARYING MGR-IDX FROM II BY 1
UNTIL MGR-IDX > REGION-MGR-MAX
    IF LETTER-IDX <= MANAGER-WORD-SIZE
        MOVE MANAGER-LETTER (LETTER-IDX)
            ... TO REGION-MGR-X (MGR-IDX)
        SET LETTER-IDX UP BY 1
    ELSE-IF REG-MGR-IDX <= REG-MGR-MAX
        MOVE REG-MGR (REG-MGR-IDX)
            ... TO REGION-MGR-X (MGR-IDX)
        SET REG-MGR-IDX UP BY 1
    ELSE
        SET MGR-IDX TO REGION-MGR-MAX
        DISPLAY 'MANAGER INDEXES OUT OF RANGE: '
        ... EXTRACT-FILE-RECORD

SET OFFICE-IDX TO 1
SEARCH OFFICE-TABLE
WHEN EXT-OFFICE = OFFICE-NAME (OFFICE-IDX)
    MOVE OFFICE-MANAGER (OFFICE-IDX) TO OFFICE-MGR

```

```
MOVE EXT-REGION TO REGION
MOVE EXT-OFFICE TO OFFICE
```

Generated Source:

```
% &AP-GEN-VER = 1719
% &AP-PGM-ID = "SUMMARY"
% &AP-GEN-DC-TARGET = "MVS"
% &AP-GEN-DB-TARGET = "VSAM"
% &AP-PROC-DIV-KYWD-SEEN = 1
% &AP-FILE-CONTROL-SEEN = 1
% &AP-SUBSCHEMA = ""
% &AP-APPLICATION-ID = "GLGAP"
% &AP-GEN-DATE = "861204"
% &AP-GEN-TIME = "17142491"
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID.                SUMMARY.
AUTHOR.                    AP-SYSTEM GENERATED.
DATE-WRITTEN.              861204.
DATE-COMPILED.            &COMPILETIME.
*
*REMARKS.
*   READS DATA.EXTRACT AND GENERATES A SUMMARY
*   REPORT BY REGION, OFFICE AND PRODUCT.
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.          &SYSTEM.
OBJECT-COMPUTER.         &SYSTEM.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EXTRACT-FILE           ASSIGN TO UT-S-EXTRACT.
    SELECT SALES-SUMMARY-FILE     ASSIGN TO UT-S-SUMMREPT.
    SELECT DEFINITION-FILE       ASSIGN TO UT-S-DEFS.
```

```
DATA DIVISION.
```

```
FILE SECTION.
```

```
FD EXTRACT-FILE
   LABEL RECORDS ARE STANDARD
   RECORDING MODE IS F
   RECORD CONTAINS 90 CHARACTERS
   BLOCK CONTAINS 0 RECORDS.
```

```

01 EXTRACT-FILE-RECORD
    02 EXT-REGION                PIC X(9).
    02 EXT-OFFICE                PIC X(15).
    02 EXT-PRODUCT              PIC X(18).
    02 EXT-SALES-DOLLARS        PIC 9(4) OCCURS 12.

FD SALES-SUMMARY-FILE
    LABEL RECORDS ARE STANDARD
    RECORDING MODE IS F
    RECORD CONTAINS 133 CHARACTERS
    BLOCK CONTAINS 0 RECORDS
    REPORT IS YEAR-END-SALES-SUMMARY.

FD DEFINITION-FILE
    LABEL RECORDS ARE STANDARD
    RECORDING MODE IS F
    RECORD CONTAINS 38 CHARACTERS
    BLOCK CONTAINS 0 RECORDS.

01 DEFINITION-RECORD.
    02 DEFINITION-TYPE          PIC X(3).
    02 DEFINITION-REGION        PIC X(29).
    02 FILLER                    PIC X(6).

01 DEFINITION-RECORD-2.
    02 FILLER                    PIC X(3).
    02 DEFINITION-OFFICE        PIC X(35).

WORKING-STORAGE SECTION.
$TP-WS-MARKER
01 JJ                            PIC S9(4) COMP SYNC VALUE ZERO.
01 II                            PIC S9(4) COMP SYNC VALUE ZERO.
01 FIRST-FLG                     PIC X(1) VALUE 'T'.

01 REGION-DEFINITIONS.
    02 REGION-TABLE OCCURS 4 TIMES INDEXED BY REGION-IDX.
        03 REGION-NAME          PIC X(9).
        03 REGION-MANAGER       PIC X(20).

01 REGION                        PIC X(9).
01 REGION-MGR.
    02 REG-MGR OCCURS 20 TIMES
        INDEXED BY REG-MGR-IDX
        PIC X(1).

01 REG-MGR-MAX                   PIC S9(4) COMP SYNC VALUE +20.

```

```

01 REGION-MGR-FIELD.
   02 REGION-MGR-X OCCURS 30 TIMES INDEXED BY MGR-IDX
                                     PIC X(1).
01 REGION-MGR-FLD REDEFINES REGION-MGR-FIELD
                                     PIC X(30).
01 REGION-MGR-MAX                       PIC S9(4) COMP SYNC VALUE +30.
01 MANAGER-WORD                          PIC X(10) VALUE 'MANAGER: '.
01 MANAGER-BY-CHAR REDEFINES MANAGER-WORD.
   02 MANAGER-LETTER OCCURS 10 TIMES
                                     INDEXED BY LETTER-IDX
                                     PIC X(1).
1  MANAGER-WORD-SIZE                      PIC S9(4) COMP SYNC VALUE +10.

01 OFFICE-DEFINITIONS.
   02 OFFICE-TABLE OCCURS 14 TIMES INDEXED BY OFFICE-IDX.
       03 OFFICE-NAME                   PIC X(15).
       03 OFFICE-MANAGER                 PIC X(20).

01 OFFICE                                 PIC X(15).
01 OFFICE-MGR                             PIC X(20).

01 QTR-1-SALES-DOLLARS                   PIC 9(5) VALUE ZERO.
01 QTR-2-SALES-DOLLARS                   PIC 9(5) VALUE ZERO.
01 QTR-3-SALES-DOLLARS                   PIC 9(5) VALUE ZERO.
01 QTR-4-SALES-DOLLARS                   PIC 9(5) VALUE ZERO.

01 YR-SALES-DOLLARS                      PIC 9(6) VALUE ZERO.
01 CURRENT-DATE-X.
   02 CURRENT-YEAR                       PIC 9(2).
   02 FILLER                              PIC X(4).

01 REPORT-YEAR.
   02 FILLER                              PIC 9(2) VALUE 19.
   02 REPORT-YEAR-X                      PIC 9(2).

REPORT SECTION.
RED YEAR-END-SALES-SUMMARY
CONTROLS ARE FINAL REGION MGR-FLD OFFICE
PAGE LIMIT IS 58 LINES
FIRST DETAIL 9
HEADING 1
FOOTING 58.

01 RH-YEAR-END-SALES-SUMMARY             TYPE IS REPORT HEADING
                                         NEXT GROUP IS NEXT PAGE.

MOCKUP LINES 1 THRU 4
LINE 25
SOURCE REPORT-YEAR                       PIC X(4).

```

01 PH-YEAR-END-SALES-SUMMARY TYPE IS PAGE HEADING.
 MOCKUP LINES 10 THRU 17
 SOURCE PAGE-COUNTER PIC ZZZZ9.
 SOURCE REGION PIC X(9).
 SOURCE REGION-MGR-FLD PIC X(30).

01 CH-REGION TYPE IS CONTROL HEADING
 REGION
 NEXT GROUP IS NEXT PAGE.

01 CH-OFFICE TYPE IS CONTROL HEADING
 OFFICE.
 MOCKUP LINES 18 THRU 20
 SOURCE OFFICE PIC X(15).
 SOURCE OFFICE-MGR PIC X(20).

01 DE-YEAR-END-SALES-SUMMARY TYPE IS DETAIL.
 MOCKUP LINE 21
 SOURCE EXT-PRODUCT PIC X(8).
 SOURCE EXT-SALES-DOLLARS (#1/3) PIC Z,ZZ9
 SOURCE QTR-1-SALES-DOLLARS PIC ZZ,ZZ9
 SOURCE EXT-SALES-DOLLARS (#4/6) PIC Z,ZZ9
 SOURCE QTR-2-SALES-DOLLARS PIC ZZ,ZZ9
 SOURCE EXT-SALES-DOLLARS (#7/9) PIC Z,ZZ9
 SOURCE QTR-3-SALES-DOLLARS PIC ZZ,ZZ9
 SOURCE EXT-SALES-DOLLARS (#10/12) PIC Z,ZZ9
 SOURCE QTR-4-SALES-DOLLARS PIC ZZ,ZZ9
 SOURCE YR-SALES-DOLLARS PIC \$\$\$\$,\$\$\$

01 PF-YEAR-END-SALES-SUMMARY TYPE IS PAGE FOOTING
 NEXT GROUP IS NEXT PAGE.
 MOCKUP LINE 32

01 RF-YEAR-END-SALES-SUMMARY TYPE IS REPORT FOOTING.
 MOCKUP LINE 38
 LINE IS 25

01 CF-FINAL TYPE IS CONTROL FOOTING
 FINAL.
 MOCKUP LINES 29 THRU 31
 SUM EXT-SALES-DOLLARS (#1/3) PIC Z,ZZ9
 SUM QTR-1-SALES-DOLLARS PIC ZZ,ZZ9
 SUM EXT-SALES-DOLLARS (#4/6) PIC Z,ZZ9
 SUM QTR-2-SALES-DOLLARS PIC ZZ,ZZ9
 SUM EXT-SALES-DOLLARS (#7/9) PIC Z,ZZ9
 SUM QTR-3-SALES-DOLLARS PIC ZZ,ZZ9
 SUM EXT-SALES-DOLLARS (#10/12) PIC Z,ZZ9
 SUM QTR-4-SALES-DOLLARS PIC ZZ,ZZ9
 SUM YR-SALES-DOLLARS PIC \$\$\$\$,\$\$\$

01 CF-REGION TYPE IS CONTROL FOOTING
REGION.

MOCKUP LINES 26 THRU 28

SOURCE REGION PIC X(9)
SUM EXT-SALES-DOLLARS (#1/3) PIC Z,ZZ9
SUM QTR-1-SALES-DOLLARS PIC ZZ,ZZ9

SUM EXT-SALES-DOLLARS (#4/6) PIC Z,ZZ9
SUM QTR-2-SALES-DOLLARS PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#7/9) PIC Z,ZZ9
SUM QTR-3-SALES-DOLLARS PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#10/12) PIC Z,ZZ9
SUM QTR-4-SALES-DOLLARS PIC ZZ,ZZ9
SUM YR-SALES-DOLLARS PIC \$\$\$\$,\$\$\$

01 CF-OFFICE TYPE IS CONTROL FOOTING
OFFICE.

MOCKUP LINES 23 THRU 25

SOURCE OFFICE PIC X(15)
SUM EXT-SALES-DOLLARS (#1/3) PIC Z,ZZ9
SUM QTR-1-SALES-DOLLARS PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#4/6) PIC Z,ZZ9
SUM QTR-2-SALES-DOLLARS PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#7/9) PIC Z,ZZ9
SUM QTR-3-SALES-DOLLARS PIC ZZ,ZZ9
SUM EXT-SALES-DOLLARS (#10/12) PIC Z,ZZ9
SUM QTR-4-SALES-DOLLARS PIC ZZ,ZZ9
SUM YR-SALES-DOLLARS PIC \$\$\$\$,\$\$\$

PROCEDURE DIVISION.

DECLARATIVES.

SUPPRESS CH-REGION SECTION.
USE BEFORE REPORTING CH-REGION

SUPPRESS CH-REGION-PARA.
IF FIRST-FLG = TRUE
SUPPRESS PRINTING

END DECLARATIVES.
ACCEPT CURRENT-DATE-X FROM DATE
MOVE CURRENT-YEAR TO REPORT-YEAR-X

PERFORM LOAD-DEFINITIONS

```

OPEN INPUT EXTRACT-FILE
OPEN OUTPUT SALES-SUMMARY-FILE

INITIATE YEAR-END-SALES-SUMMARY
MOVE ZERO TO PAGE-COUNTER

REPEAT
  READ EXTRACT-FILE
UNTIL AT END ON EXTRACT-FILE
  IF EXT-OFFICE NOT = OFFICE
    PERFORM LOCATE-MANAGERS

    ADD EXT-SALES-DOLLARS (1) TO QTR-1-SALES-DOLLARS
    ADD EXT-SALES-DOLLARS (2) TO QTR-1-SALES-DOLLARS
    ADD EXT-SALES-DOLLARS (3) TO QTR-1-SALES-DOLLARS

    ADD EXT-SALES-DOLLARS (4) TO QTR-2-SALES-DOLLARS
    ADD EXT-SALES-DOLLARS (5) TO QTR-2-SALES-DOLLARS
    ADD EXT-SALES-DOLLARS (6) TO QTR-2-SALES-DOLLARS

    ADD EXT-SALES-DOLLARS (7) TO QTR-3-SALES-DOLLARS
    ADD EXT-SALES-DOLLARS (8) TO QTR-3-SALES-DOLLARS
    ADD EXT-SALES-DOLLARS (9) TO QTR-3-SALES-DOLLARS

    ADD EXT-SALES-DOLLARS (10) TO QTR-4-SALES-DOLLARS
    ADD EXT-SALES-DOLLARS (11) TO QTR-4-SALES-DOLLARS
    ADD EXT-SALES-DOLLARS (12) TO QTR-4-SALES-DOLLARS

  REPEAT VARYING II FROM 1 BY 1
  UNTIL II > 12
    ADD EXT-SALES-DOLLARS (II) TO YR-SALES-DOLLARS

GENERATE DE-YEAR-END-SALES-SUMMARY

MOVE FALSE TO FIRST-FLG

MOVE ZEROES TO QTR-1-SALES-DOLLARS
... QTR-2-SALES-DOLLARS QTR-3-SALES-DOLLARS
... QTR-4-SALES-DOLLARS YR-SALES-DOLLARS

TERMINATE YEAR-END-SALES-SUMMARY

CLOSE EXTRACT-FILE SALES-SUMMARY-FILE

LOAD-DEFINITIONS.
SET REGION-IDX TO 1

```

```

SET REGION-IDX DOWN BY 1
SET OFFICE-IDX TO 1
SET OFFICE-IDX DOWN BY 1

OPEN INPUT DEFINITION-FILE

REPEAT
    READ DEFINITION-FILE
UNTIL AT END ON DEFINITION-FILE

    IF DEFINITION-TYPE = 'REG'
        SET REGION-IDX UP BY 1
        MOVE DEFINITION-REGION
                                TO REGION-TABLE (REGION-IDX)

    ELSE-IF DEFINITION-TYPE = 'OFF'
        SET OFFICE-IDX UP BY 1
        MOVE DEFINITION-OFFICE
                                TO OFFICE-TABLE (OFFICE-IDX)

CLOSE DEFINITION-FILE

LOCATE-MANAGERS.
    SET REGION-IDX TO 1
    SEARCH REGION-TABLE
    WHEN EXT-REGION = REGION-NAME (REGION-IDX)
        MOVE REGION-MANAGER (REGION-IDX) TO REGION-MGR
    MOVE SPACES TO REGION-MGR-FLD

    SET REG-MGR-IDX TO REG-MGR-MAX
    WHILE REG-MGR (REG-MGR-IDX) = SPACE
        ... AND REG-MGR-IDX > ZERO
            SET REG-MGR-IDX DOWN BY 1
    SET JJ TO REG-MGR-IDX

    COMPUTE II =
        ... (REGION-MGR-MAX - MANAGER-WORD-SIZE - JJ) / 2
    ADD 1 TO II

    IF II <= ZERO
        MOVE 1 TO II
    SET REG-MGR-IDX TO 1
    SET LETTER-IDX TO 1

    REPEAT VARYING MGR-IDX FROM II BY 1
    UNTIL MGR-IDX > REGION-MGR-MAX
        IF LETTER-IDX <= MANAGER-WORD-SIZE

```

```

        MOVE MANAGER-LETTER (LETTER-IDX)
        ... TO REGION-MGR-X (MGR-IDX)
        SET LETTER-IDX UP BY 1
    ELSE-IF REG-MGR-IDX <= REG-MGR-MAX
        MOVE REG-MGR (REG-MGR-IDX)
        ... TO REGION-MGR-X (MGR-IDX)
        SET REG-MGR-IDX UP BY 1
    ELSE
        SET MGR-IDX TO REGION-MGR-MAX
        DISPLAY 'MANAGER INDEXES OUT OF RANGE: '
        ... EXTRACT-FILE-RECORD

    SET OFFICE-IDX TO 1
    SEARCH OFFICE-TABLE
    WHEN EXT-OFFICE = OFFICE-NAME (OFFICE-IDX)
        MOVE OFFICE-MANAGER (OFFICE-IDX) TO OFFICE-MGR

    MOVE EXT-REGION TO REGION
    MOVE EXT-OFFICE TO OFFICE

```

Report Writer Structures

Description: After you paint your report mock-up in the Report Painter, you must use APS Report Writer structures to code report logic. These structures let you automatically page the report, define headers and footers, calculate field values, test and execute control and page breaks, generate multiple reports, and generate all logic necessary to map fields between reports and databases or files.

Code Report Writer structures in the Program Painter for report programs.

List of Structures:	<i>IO and FD</i>	Name the input and output files.
	<i>RED</i>	Add a Report Section to your program.
	<i>CODE</i>	Specify a 2-character literal that identifies each print line with a specific report.
	<i>CONTROL</i>	Identify data items that cause control breaks.
	<i>WRITE ROUTINE</i>	Override a standard COBOL WRITE statement and execute your own routine.

<i>PAGE LIMIT</i>	Define the report format, such as the number of lines per page and where report lines appear on the page.
<i>MOCK</i>	Identify the report mock-up.
<i>01 and TYPE</i>	Describe function, format, and characteristics of each report line.
<i>MOCKUP LINES</i>	Map the report mock-up lines to the lines on the printed report.
<i>OVERPRINT</i>	Highlight or underscore the lines identified in the <i>MOCKUP LINES</i> clause.
<i>SOURCE</i>	Map the report mock-up fields to the output fields on the printed report.
<i>VALUE (Report Writer)</i>	Designate a literal value to print for the field each time the line prints.
<i>REFERENCE</i>	Identify a non-printing data field for summing in a control break.
<i>SUM</i>	Establish a sum accumulator for a corresponding <i>SOURCE</i> or <i>REFERENCE</i> data field, and print the total in a control break.
<i>INITIATE</i>	Generate multiple <i>SOURCE</i> and <i>SUM</i> statements for suffixed data items or array elements with minimal coding.
<i>GENERATE</i>	Open report files and initialize page, line, and sum counters and accumulators.
<i>USE BEFORE REPORTING</i>	Generate and print all the report lines.
<i>TERMINATE</i>	Specify additional processing for a report group prior to printing.

Syntax Rules: Code Report Writer structures in the Program Painter, associating the structures with keywords, as shown in the sample skeletal Report Writer program below.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50
  IO      Input/Output statements
  .
  .
  FD      Input FD clause
  .
  .
```

```

01      Input record description

FD      Output FD clause
. . .
01      Output record description

RED     reportfilename
        CODE clause
        CONTROL clause
        WRITE ROUTINE clause
        PAGE LIMIT nn LINE
           FIRST DETAIL linenumber
           LAST DETAIL linenumber
           FOOTING linenumber.

mock mockupreportname

01      TYPE IS REPORT HEADING /*for report header
        MOCKUP LINES clause
        OVERPRINT clause
        SOURCE clause or VALUE clause

01      TYPE PAGE HEADING      /*for page header
        MOCKUP LINES clause
        SOURCE clause or VALUE clause

01      TYPE CONTROL HEADING /*for control header
        MOCKUP LINES clause
        SOURCE clause or VALUE clause

01      TYPE DETAIL           /*for detail lines
        MOCKUP LINES clause
        SOURCE clause or VALUE clause
        REFERENCE clause

01      TYPE CONTROL FOOTING /*for control break
        MOCKUP clause
        SOURCE clause or VALUE clause
        SUM clause

NTRY
.
INITIATE statement
.
GENERATE statement
.
TERMINATE statement
.

```

Reports, Application-Generated

Description: APS provides a set of reports that help you understand your application and its various components. Use these reports as you develop an application to determine the status of your work and the tasks left to complete. Some reports help you to troubleshoot problems in an application that you are developing, or to determine the impact of a proposed change. Others help you to verify the results of your work. Once you have fully implemented an application, use the APS reports to document it so that developers who later maintain or enhance the application can easily understand it in detail.

You can produce reports on an entire application, on selected components, or on selected members of components. You can produce reports from the Report Generator, Painter Menu, Application Painter, or Documentation Facility, as follows:

List of Reports:	Report	Available In
	Application Definition (AP01) lists and describes all components of an application except the scenario prototype.	Painter Menu Application Painter Report Generator
	Component List (MS01) catalogs and totals the components for each painter.	Documentation Facility
	Data Structure Definition (DS01) lists and describes structures that you create in the Data Structure Painter.	Painter Menu Application Painter Report Generator
	DDIFILE (DB01) describes the contents of the file that contains information about your database, formatted to APS specifications.	Documentation Facility
	Entity Content (MS02) lists summary information for each application component.	Documentation Facility
	Entity Cross Reference (MD01) cross references and totals application components.	Documentation Facility
	Entity Parts List (EN01) catalogs selected parts of one or more application components.	Documentation Facility

Report	Available In
Entity Search Utility (GS01) lets you create reports on application components that meet the selection criteria that you specify.	Documentation Facility
Entity Use (EN02) lists components that copy, include, or otherwise use the target component.	Documentation Facility
Field/Screen Cross Reference (SC02) lists application screens along with their I/O and text fields.	Documentation Facility
Macro/Program Cross-Reference (MC01) lists macros and the programs that use them.	Documentation Facility
Mock-Up (RP01) lists and displays report mock-ups as painted in the Report Painter.	Painter Menu Application Painter Report Generator
Program DB/DC (PG02) lists the screens and the subschemas or PSBs used by a program.	Documentation Facility
Program Definition (PG01) provides a printout of programs created in APS.	Painter Menu Application Painter Report Generator
Scenario Definition (CN01) describes components created in the Scenario Prototype Painter.	Painter Menu Application Painter Report Generator
Screen Hardcopy/Field Attribute (SC01) displays the components of a screen as painted in the Screen Painter as well as field attribute and field edit information.	Painter Menu Application Painter Report Generator
See these individual report descriptions for details about the content and format of the information provided.	

Reserved Words

The following words are reserved for APS use.

%	BIND	CONTINUE	DISCONNECT
@	BLANK	CONTROL[S]	DISPLAY
&	BLOCK	COPY	DIVIDE
&&	BYTES	COPYLIB	DIVISION
--	CA	CPERFORM	\$DLG-
\$	CALL	DATA	DPAR
+	CANCEL	DATA-NAME	DS
++	CARDIN	\$DB-	EDIT-FLAGS
<+	CBL	DB-CLOSE	EJECT
</	CF	DB-ERASE	ELSE
=	CH	DB-IF	ELSE-IF
<	CHANGE	DB-MODIFY	END
<<	CHARACTERS	DB-OBTAIN	END-OBTAIN
<*	\$CIC-	DB-OPEN	ENTER
/*01	CICS	DB-ROLLBACK	ENTRY
ACCEPT	CLEAR	DB-STORE	*EOF*
ACCESS	CLOSE	DC	ERASE
ADD	COBMESS	\$DDI-	ERROR
ALTERNATE	COBIIMES	DDISYMB	ESCAPE
APPLY	CODE	DE	EVALUATE
\$APS	CODE-SET	DEBUG	EXAMINE
APSMACS	COLUMN	DECL	EXEC
APSSRC	COMMIT	DECLARATIVES	EXHIBIT
ARE	COMP	&DEFINE[D]	EXIT
ASSIGN	&COMPILETIME	&DEFVAL	FALSE
ATTR	COMPUTATIONAL	DELETE	FD
AUXOUT	COMPUTE	DEPENDING	FILE
BASIS	CONNECT	DESTINATION	FILE-ID
BEFORE	CONTAINS	DETAIL	FILE-LIMITS

FILLER	LAST	NTRY	PROC
FIND	LEADING	NUMBER	PROCEDURE
FINISH	&LENGTH	&NUMERIC	PROCESSING
FIRST	LIB1[IN]	OBTAIN	QUOTE[S]
FOOTING	LIB2[IN]	OBTAIN-BY-KEY	RDREAD
FRFM	LIB3[IN]	OBTAIN-BY-SEARCH	READ
GENERATE	LIMIT	OBTAIN-NEXT	READY
GET	LIMITS	OBTAIN-PREV	REC
GO	LINE[S]	OBTAIN-REL	RECEIVE
GOBACK	LINE-COUNTER	OBTAIN-REL-BY-KEY	RECORD[S]
GROUP	LINK	OCCURS	RECORDING
HEADING	LINKAGE	OF	RD
HIGH-VALUE	LK	OMITTED	RED
HIGH-VALUES	LOW-VALUE[S]	ON	REDEFINES
IDM-	\$MACRO-	OPEN	REFERENCE
IF	MACRO	OPT	RELEASE
\$IM-	MAININ	ORGANIZATION	REM
\$IMS-	\$MDB-	OUTPUT	RENAMES
IN	\$MDC-	OVERPRINT	REPEAT
&INDEX	MERGE	PAGE	REPORT[S]
INDEX[ED]	MOCK	PAGE-COUNTER	REPORTING
INDICATE	MOCKUP	PARA	RERUN
INITIATE	MODE	&PARSE	RESERVE
INPUT	MODIFY	PASS	RESET
INPUT-OUTPUT	MOD-NAME	PASSWORD	RETURN
INSERT	MOVE	PERFORM	REWRITE
INSPECT	MSG-SW	PF	RF
IO	MULTIPLY	PFKEY-VALUE	RH
IS	NARROW	PH	RIGHT
JUST	NOMINAL	PIC	ROLLBACK
JUSTIFIED	NOT	PICTURE	SAME
KEEP	NOTE	PLUSPOSTSOUT	\$SC-
LABEL	NEXT	PRIVIN	SC-CLEAR

SCELIB	STANDARD	SYRP	USEUSERMACS
\$SCP-	START	SYSDBOUT	USERNAME
\$SCR-	STATUS	SYSIN	VALUE[S]
SCRNLST	STOP	SYSOUT	\$VS-
SCRSYMB	STORE	SYSMSG	\$VSAM-
SD	STRING	SYWS	WHEN
SPACE[S]	STUB	TERM	WHILE
SPNM	&SUBSTR	TERMINATE	WITH
SEARCH	SUBTRACT	TEXT	WORK1
SECTION	SUM	TIMES	WORK2
SEEK	SUPRASUPPRESS	\$TP-	WORK3
SELECT	SYBT	TRACE	WORK4
SEND	SYDD	TRAILING	WORK5
SERVICE	SYEN	TRANCODE-AREA	WORK6
SET	SYFD	TRANSFORM	WORK7
SIGN	SYLK	TRUE	WORK8
SKIP1	SYIO	TYPE	WORK9
SKIP2	SYM1	UNSTRING	WORKING-STORAGE
SKIP3	SYM2	UNTIL	WRITE
SORT	SYMBOLIC	UPDATE	WS
SOURCE	SYNC	UPON	XCTL
SQL	SYNCHRONIZED	USAGE	ZERO[S][ES]

RESET-PFKEY

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS, IMS DC, ISPF Dialog, and ISPF Prototyping targets

Description: Simulate screen invocation.

Syntax: [TP-]RESET-PFKEY *keyvalue*

Parameters: Valid *keyvalues* are the following.

Value	CICS	IMS DC	ISPF Dialog	ISPF Prototyping
PF0	Y	Y	N	N
PF00	Y	Y	N	N
PF01 thru PF24	Y	Y	Y	Y
ENTER	Y	Y	Y	Y
ENTER-KEY	Y	Y	Y	Y
PA1	Y	N	N	N
PA2	Y	N	N	N
PA3	Y	N	N	N
CLEAR-KEY	Y	N	N	N
CLEAR	Y	N	N	N
NO-KEY	N	Y	N	N
NONE	N	Y	N	N
PEN	Y	N	N	N
PFXX	N	N	N	Y
PFZZ	N	N	N	Y
TRIG	Y	N	N	N
OPID	Y	N	N	N
MSRE	Y	N	N	N
STRF	Y	N	N	N
PAGE-DOWN	N	N	N	N
PAGE-UP	N	N	N	N
ROLL-UP	N	N	N	N
ROLL-DOWN	N	N	N	N
PRINT-KEY	N	N	N	N
PRINT	N	N	N	N
HELP-KEY	N	N	N	N
HELP	N	N	N	N
HOME-KEY	N	N	N	N
HOME	N	N	N	N

Comment: ISPF Dialog

To use this call, set the PF Key Option field to P on the ISPF Panel Options screen.

Example: Simulate pressing the ENTER key.

```

/* Move Commarea key to key field
MOVE CA-PASSED-KEY TO SCRN-KEY-FLD
/* Move Q to FUNCTION field
MOVE 'Q' TO SCRN-FUNCTION-FLD
/* Simulate pressing of ENTER key
/* with RESET-PFKEY call
RESET-PFKEY ENTER
/* Perform processing logic
TP-PERFORM APS-USER-CODE-PARA

```

S-COBOL Structures

Description: An APS program can be coded in APS Structured COBOL (S-COBOL) language structures, either in combination with or instead of COBOL statements. S-COBOL structures are procedural, and extend the power of native COBOL, yet simplify and shorten the amount of code you must write.

You can write a complete APS program in S-COBOL. There are no differences between S-COBOL and batch COBOL in the Identification, Environment, and Data Divisions. The major differences exist in the Procedure Division.

S-COBOL reserved words are the same as ANSI COBOL, plus the following. For a complete list of all APS reserved words and symbols, see *Reserved Words*.

ALWAYS	ELSE-IF	ESCAPE
EVALUATE	FALSE	FALSX
NEVER	REPEAT	SAGE-TRACE-FLAG
TRUE	TRUX	UNTIL
USERNAME (IMS only)	WHILE	

You code the S-COBOL structures in paragraphs and statement blocks. A paragraph is denoted by the keyword `PARA` and a paragraph name, followed by clauses or a statement block. Within a paragraph, indentation determines the exact positioning of statements. A paragraph ends with the next appearance of any Procedure Division keyword.

Indentation determines how the statements execute. For example, when you indent a statement under a conditional statement, the indentation tells APS that this statement is subordinate to the condition.

APS reads an S-COBOL program from top to bottom. The first paragraph performs all other paragraphs. Within paragraphs and statement blocks, statements execute sequentially until conditional statements, `PERFORMs`, or `CALLs` modify the sequence of operations. Indentation controls the logical sequence in which lines of source code execute.

Code S-COBOL structures in the Program Painter for batch, report, or complex online programs, or in the Online Express Specification Editor for an Express program.

List of Structures: The following are the S-COBOL structures.

Verbs

<i>ENTRY</i>	Establish entry point for subprogram.
<i>ESCAPE</i>	Exit from the current paragraph.
<i>EVALUATE</i>	Evaluate one or more conditions; program a decision table.
<i>EXIT PROGRAM</i>	End the execution sequence.
<i>PERFORM</i>	Execute a particular paragraph or section, with or without arguments.
<i>REPEAT</i>	Establish a loop for testing.
<i>SEARCH</i>	Establish looping or conditionals for each <code>WHEN</code> condition.
<i>STOP RUN</i>	Return control to the operating system.
<i>TRUE/FALSE</i>	Establish true and false flags; test the flags.
<i>UNTIL/WHILE</i>	Form a loop with a test.
<i>USERNAME</i>	Title a procedure generated by the APS Precompiler.

Conditionals

<i>AT END/INVALID KEY</i>	Test for end-of-file or invalid key condition.
<i>EVALUATE</i>	Evaluate one or more conditions; program a decision table.
<i>IF/ELSE-IF/ELSE</i>	Evaluate one or more conditions.
<i>SEARCH</i>	Establish looping or conditionals for each WHEN condition.
<i>TRUE/FALSE</i>	Establish true and false flags; test the flags.

Error Handling

<i>AT END/INVALID KEY</i>	Test for end-of-file or invalid key condition.
<i>TRUE, FALSE, ALWAYS, NEVER</i>	Use these APS-supplied flags for testing.
<i>TRUE/FALSE</i>	Establish true and false flags; test the flags.
<i>SAGE-TRACE-FLAG</i>	Debug with the Trace facility.

Flags

<i>TRUE, FALSE, ALWAYS, NEVER</i>	Use these APS-supplied flags for testing.
<i>SAGE-TRACE-FLAG</i>	Debug with the Trace facility.

Looping

<i>REPEAT</i>	Establish a loop for testing.
<i>UNTIL/WHILE</i>	Form a loop with a test.

Transferring Program Control

<i>ENTRY</i>	Establish entry point for subprogram.
<i>ESCAPE</i>	Exit from the current paragraph.
<i>EXIT PROGRAM</i>	End the execution sequence.
<i>STOP RUN</i>	Return control to the operating system.

Syntax Rules: Observe the following rules and conventions when coding an S-COBOL program.

- When coding paragraph names, at least one character within the first 24 characters must be unique for each paragraph in the program.

- Code only one verb per source code line.
- Do not code punctuation in the Procedure Division. It is unnecessary and therefore disregarded.
- Code punctuation within the Identification, Environment, and Data Divisions that conforms to COBOL rules.
- Code structures with consistent indentation. The level of indentation can vary with each new statement block; however, it cannot vary within a statement block of code. We recommend using four additional spaces for each new level of indentation.
- Do not code the following structures.

```

PERFORM ... THRU statement

PERFORM paragraphname(arg1, arg2, ..., argN)
... UNTIL condition

GO TO statement

SORT|MERGE THRU ... sectionname with INPUT PROCEDURE or
      OUTPUT PROCEDURE.

NEXT SENTENCE

```

- Avoid using the double-hyphen (--). The S-COBOL translator frequently creates procedures and flags required for generating ANSI COBOL that are unnecessary and sometimes invalid in S-COBOL. Generated names for these procedures or flags always contain a double-hyphen to avoid conflict with programmed names. Under IMS, the USERNAME parameter can change double hyphens to single hyphens at S-COBOL processing time.
- Continue a structure on subsequent lines by coding an ellipsis followed by a space (...).
- Continue a non-numeric literal by splitting the literal and concatenating the parts, which APS then treats as separate literals. Concatenate non-numeric literals by separating them with an ellipsis, space, two ampersands, and space (... &&), for example,


```

-KYWD- 12--*--20---*-----30-----*---40
        REGISTER = " THIS IS A VERY, "
        ... && "VERY LONG LITERAL"

```
- Enclose literals with quotation marks. S-COBOL allows both single and double quotation marks within the same program.

- Use any of the following relational operators to make comparisons.

Type of Comparison	Relational Operator
Greater than	IS GREATER THAN IS >
Not greater than	IS NOT GREATER THAN IS NOT > IS <=
Less than	IS LESS THAN IS <
Not less than	IS NOT LESS THAN IS NOT < IS >=
Equal to	IS EQUAL TO IS =
Not equal to	IS NOT EQUAL TO IS NOT =
Greater than or equal to	>=
Less than or equal to	<=

- Create compound conditions by using ANDs or ORs.
- Optionally, use a simplified syntax

Abbreviated Syntax	S-COBOL/COBOL Equivalent
A = B	MOVE B TO A
A B = C	MOVE C TO A B
A = B + C	COMPUTE A = B + C
B = B + C	COMPUTE B = B + C
A = B - C	COMPUTE A = B - C
A = B * C	COMPUTE A = B * C
A = B / C	COMPUTE A = B / C
A = B + (C * D)	COMPUTE A = B + (C * D)
A <+ B	COMPUTE A = A + B
A </ B	COMPUTE A = A / B
A << B	MOVE B TO A
A <* B	COMPUTE A = A * B

- Use the above simplified syntax with any of these verbs - ADD, COMPUTE, DIVIDE, MOVE, MULTIPLY, SUBTRACT

- The following syntax is ambiguous, so results are unpredictable:

```
IF condition = condition
ELSE-IF condition = condition
    ELSE-IF condition = condition
```

To achieve correct results, code:

```
IF condition = condition
ELSE-IF condition = condition
ELSE-IF condition = condition
```

or:

```
IF condition = condition
ELSE-IF condition = condition
    IF condition = condition
    ELSE-IF condition = condition
```

Example: The following example demonstrates control logic without interrupting the flow or readability of the program with a GO TO. Program control is handled entirely by S-COBOL indentation.

```
-LINE- -KYWD- 12--*--20---*-----30----*---40---*-----50---*--
001030  PARA   PAYROLL-CALCULATION /*PARAGRAPH NAME
001040          IF EMPLOYEE-TYPE = 'HRLY'
001050              IF HOURS-WORKED > 40
001060                  HOURS-BASE = HOURS-WORKED * 2
001070                  HOURS-BASE = HOURS-BASE - 40
001080              ELSE
001090                  HOURS-BASE = HOURS-WORKED
001100                  GROSS-PAY = HOURS-BASE * HOURS-RATE
001110              ELSE
001120                  GROSS-PAY = EMPLOYEE-SALARY
```

Because IF conditions generally require an alternative action, you generally code an ELSE-IF or ELSE statement at the same level of indentation as the IF. In the previous example, if EMPLOYEE-TYPE is not equal to HRLY (line 1040), control passes to the corresponding ELSE statement on line 1110. On the other hand, if EMPLOYEE-TYPE is equal to HRLY, lines 1050-1100 execute. In this case, if HOURS-WORKED is not greater than 40, control passes to the ELSE statement at the same indentation (line 1080). Thus, lines 1060-1070 execute only for hourly employees who have worked over 40 hours, while line 1090 executes for hourly employees who have worked 40 hours or less.

Line 1100 executes for all hourly employees, because these lines are at the same level as the preceding IF/ELSE. Line 1120 executes for employees who are not hourly.

Related Topics: See... **For other information about coding S-COBOL programs...**
 See *PARA and Paragraphs* Coding statement blocks
 See *Limits* S-COBOL limits

Scenario Definition Report (CN01)

Category: APS-generated report (see *Application Reports*)

Description: The Scenario Definition Report displays the Scenario Painter components as they are painted, along with descriptive information such as comments and screen title, as well as each screen in the scenario as painted. Use this report to document scenarios for end users, so they can help you determine whether your scenarios cover all of the required cases.

Comment: Produce the Scenario Definition Report from the Report Generator by selecting Actions and then the set of members you want. Or, enter **1, 2, or 3** in the **Option** field. If you enter **3**, enter a member name or range of member names for selecting report data and then enter **cn** in the **Library** field. Press Enter to submit a job to produce the report.

Example:

```

REPORT CODE: CN01                APS APPLICATION PAINTER                PAGE      1
                                SCENARIO DEFINITION REPORT          01/18/92 15:00
                                CLSAPS .CLS2

SELECTION CRITERIA:
    TDDEM01
*****
SCENARIO: TDDEM01                CREATED: 10/17/90
TITLE:                            UPDATED: 10/17/90
*****

LINE      SCREEN      SCREEN TITLE                        USER COMMENT
-----
0001      TDME      CUSTOMER ORDER MAIN MENU          MENU SELECTS ONLY
0002      TDCM      CUSTOMER MAINTENANCE              BROWSE ONLY
0003      TDPL      PARTS INVENTORY LIST
    
```

0004	TDOM	ORDER RECORD MAINTENANCE	
0005	TDOT	ORDER COST TOTALS	SUMMARY SCREEN
0006	TDOJ	CUSTOMER ORDERS INQUIRY	BROWSE ONLY
0007	TDOU	GIFT CERTIFICATES	
0008	TDCS	ORDERS DELIVERY LIST	
0009	TDPF	PART LIST SELECTIVE INFO	
0010	TDPM	PART RECORD MAINTENANCE	UPDATE

Screen Hardcopy/Field Attribute Report (SC01)

Category: APS-generated report (see *Application Reports*)

Description: The Screen Hardcopy/Field Attribute Report provides a mock-up of the screens you select as they are painted. In addition, the report includes sections that describe field attributes and field edits.

The Field Attribute portion lists the attribute values for each field in the order it appears on the screen. The report indicates default values with periods (.). The Occ(urrence) column shows the number of field occurrences within a repeated block. The end of the report shows the total number of screens on the report.

The Field Edits section lists the field edits applied to the fields.

- Comments:**
- Produce the Screen Hardcopy/Field Attribute Report from the Report Generator, Painter Menu, or Application Painter.
 - You can then set or change generation options for the report. To do so, access the Report Options screen in one of the following ways.
 - From the Report Generator screen, select **Options Report Options**, or enter **4** in the **Option** field.
 - From any APS screen, select **Options Report Options** from the action bar, or enter **opt** in the **Command** or **Options** field. From the APS Options Menu that displays, select **Actions Report Options** from the action bar, or enter option **4** in the **Options** field.

Complete the fields on the Report Options screen.

Field	Values
Left Justify Report	Y Print the Screen Hardcopy Report (SC01) starting in column 1 to fit an 8.5 x 11 inch page.
	N Default. Center the report.
Field Attributes Report	Y Default. Print the Field Attributes Report with the Screen Hardcopy Report.
	N Do not print the Field Attributes Report.
Field Edits Report	Y Default. Print the Field Edits Report with the Screen Hardcopy Report.
	N Do not print the Field Edits Report.

Example:

```
REPORT CODE: REPT                APS 1.8  ENTITY REPORT FACILITY                PAGE      1
          CLSAPS.CLS2                                01/18/92          15:04
REPORT CRITERIA:
```

```
  ALL SCREENS IN THE APPLICATION : TDDEMO
*****
  LIBRARY      ENTITY
  TYPE         NAME          STATUS      REMARKS
  -----
  AP           TDDEMO        REPORTED
  SC           TDCM          REPORTED
  SC           TDCS          REPORTED
  SC           TDME          REPORTED
  SC           TDOJ          REPORTED
  SC           TDOM          REPORTED
  SC           TDOT          REPORTED
  SC           TDOU          REPORTED
  SC           TDPF          REPORTED
  SC           TDPL          REPORTED
  SC           TDPM          REPORTED
```

```
REPORT CODE: AP01                APS APPLICATION PAINTER                PAGE      1
          CLSAPS.CLS2                                01/18/92          15:04
APPLICATION DEFINITION REPORT
```

```
SELECTION CRITERIA:
  TDDEMO
*****
APPLICATION: TDDEMO                CREATED: 02/15/90
TITLE      :                       UPDATED: 08/30/90
AUTHOR    : CLSTR1                DC TARGET: CICS
                                          DB TARGET: VSAM
*****
```

-LINE-	PROGRAMS	SCREENS	IO	REPORTS	DATA STR	TY	SBSC/PSB	USERMACS	LOC
000001	TDME	TDME	IO						
000002	TDCM	TDCM	IO				Tddb2		
000003	TDPL	TDPL	IO				Tddb2		

```

000004   TDOM      TDOM      IO          TDOB2
000005   TDOJ      TDOJ      IO          TDOB2
000006   TDOJ      TDOJ      IO          TDOB2
000007   TDOU      TDOU      IO          TDOB2
000008   TDCS      TDCS      IO          TDOB2
000009   TDPF      TDPF      IO          TDOB2
000010   TDFM      TDFM      IO          TDOB2
REPORT CODE: SC01                APS SCREEN PAINTER                PAGE      1
                                  SCREEN HARDCOPY REPORT            01/18/92   15:04
                                  CLSAPS.CLS2

```

```

SELECTION CRITERIA:
  SCREEN-NAME = TDCM
*****
ENTITY:   TDCM                      CREATED:   09/13/89
TITLE:                                         UPDATED:  03/20/90
*****
-+---1-+---2-+---3-+---4-+---5-+---6-+---7-+---8
          CUSTOMER ORDER ENTRY SYSTEM

```

APS DEVELOPMENT CENTER
CUSTOMER RECORD MAINTENANCE

```

FUNCTION =====> X (Q-QUERY U-UPDATE A-ADD D-DELETE)

CUSTOMER NUMBER ==> XXXXXX          XXXXXXXX
CUSTOMER NAME   ==> XXXXXXXXXXXXXXXXXXXX
CUSTOMER ADDRESS ==> XXXXXXXXXXXXXXXXXXXX
CUSTOMER CITY   ==> XXXXXXXXXXXXXXXXXXXX
CUSTOMER ZIP    ==> XXXXXXXXXX

```

Enter CUSTOMER NUMBER TO QUERY A RECORD

PF3 = MAIN MENU

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
---+---1-+---2-+---3-+---4-+---5-+---6-+---7-+---8
*****
REPORT CODE: SC01                APS SCREEN PAINTER                PAGE      1
                                  FIELD ATTRIBUTE REPORT            01/18/91   15:04
                                  CLSAPS.CLS2
SELECTION CRITERIA:
  SCREEN-NAME = TDCM

```

```

*****
ENTITY:   TDCM                      CREATED:   09/13/89
TITLE:                                         UPDATED:  03/20/90
*****

```

FIELD	OCC	ROW	COL	TYPE	INTEN	MDT	NUM	DET	MASK	MOD	EDIT	HI-	INIT
											COLOR	LITE	CURS
FUNCTION	7	29	1	UNPR	BRIGHT	T . .
CUSTOMER-NO	9	29	6	UNPR	BRIGHT
SAVEKEY	9	49	7	PROT	DARK
CUSTOMER-NAME	10	29	20	UNPR	BRIGHT
CUSTOMER-ADDR	11	29	20	UNPR	BRIGHT
CUSTOMER-CITY	12	29	20	UNPR	BRIGHT
CUSTOMER-ZIP	13	29	9	UNPR	BRIGHT
MESSAGE	24	2	79	PROT	BRIGHT

```

REPORT CODE: ED01                APS SCREEN PAINTER                PAGE      1
                                  FIELD EDIT REPORT                01/18/92   15:04
                                  CLSAPS.CLS2

```

Reference

SELECTION CRITERIA:
 SCREEN-NAME = TDCM

```
*****
ENTITY:  TDCM                                CREATED: 09/13/89
TITLE:                                     UPDATED: 03/20/90
*****
```

DEFAULT SCREEN LEVEL ERROR PROCESSING:

```
-----
SYMSG FIELD: MESSAGE
ERROR ATTRIBUTING: POS+BRT
ERROR MESSAGE      : FIELD AT CURSOR IS IN ERROR
REQUIRED FIELD MESSAGE: FIELD AT CURSOR IS REQUIRED
```

CONDITIONS TO BYPASS INPUT EDITING:

```
-----
BYPASS INPUT EDITS IF
FUNCTION = 'Q'
```

```
BYPASS INPUT EDITS IF
PF01 ==>    PF09 ==>    PF17 ==>    PA01 ==>
PF02 ==>    PF10 ==>    PF18 ==>    PA02 ==>
PF03 ==> S  PF11 ==>    PF19 ==>    CLEAR ==>
PF04 ==>    PF12 ==>    PF20 ==>    Enter ==>
PF05 ==>    PF13 ==>    PF21 ==>
PF06 ==>    PF14 ==>    PF22 ==>
PF07 ==>    PF15 ==>    PF23 ==>
PF08 ==>    PF16 ==>    PF24 ==>
```

FIELD

```
-----
CUSTOMER-NO      LEN:   6  ROW:   9  COL:  29  INTERNAL PICTURE: X(06)
```

```
INPUT EDITING:
REQUIRED
NUMERIC TEST
```

```
ERROR PROCESSING:
ERROR ATTRIBUTING: POS+BRT
ERROR MESSAGE      : YOU MUST Enter A NUMERIC VALUE
REQUIRED FIELD MESSAGE: YOU MUST Enter A CUSTOMER NUMBER
```

```
REPORT CODE: ED01      APS SCREEN PAINTER      PAGE    2
                      FIELD EDIT REPORT      01/18/92    15:04
                      CLSAPS.CLS2
```

SELECTION CRITERIA:
 SCREEN-NAME = TDCM

```
CUSTOMER-NAME  LEN:  20  ROW:  10  COL:  29  INTERNAL PICTURE: X(20)
```

```
INPUT EDITING:
REQUIRED
INPUT MASK: ADDDDDDDDDDDDDDDDDD
```

```
ERROR PROCESSING:
ERROR ATTRIBUTING: POS+BRT
ERROR MESSAGE      : CUSTOMER NAME MUST NOT BE NUMERIC
REQUIRED FIELD MESSAGE: YOU MUST Enter CUSTOMER NAME
```

```
CUSTOMER-ADDR  LEN:  20  ROW:  11  COL:  29  INTERNAL PICTURE: X(20)
```

```
INPUT EDITING:
REQUIRED
```

```

ERROR PROCESSING:
  ERROR ATTRIBUTING: POS+BRT
  REQUIRED FIELD MESSAGE: CUSTOMER ADDRESS MUST BE ENTERED

```

```
CUSTOMER-CITY  LEN:  20  ROW:  12  COL:  29  INTERNAL PICTURE: X(20)
```

```

INPUT EDITING:
  REQUIRED

```

```

ERROR PROCESSING:
  ERROR ATTRIBUTING: POS+BRT
  REQUIRED FIELD MESSAGE: YOU MUST Enter THE CUSTOMER CITY

```

```
CUSTOMER-ZIP  LEN:   9  ROW:  13  COL:  29  INTERNAL PICTURE: X(09)
```

```

INPUT EDITING:
  REQUIRED

```

```

ERROR PROCESSING:
  ERROR ATTRIBUTING: POS+BRT
  REQUIRED FIELD MESSAGE: YOU MUST Enter THE POSTAL CODE

```

Screen Redefinition

Description: During screen generation, APS automatically generates record definitions describing the data layout and format of a screen. You can redefine the entire generated screen record, selected tables in the screen, or selected fields in the screen.

To do so, you set a flag to enable redefinition, and create user macros to generate the redefinition code. APS provides macro name suffixes that the flags recognize and invoke automatically.

To redefine multiple screens, see *SCRNLIST*.

Procedure: 1 To set a flag to enable redefinition, go to the APS CNTL file GENSymb2 and set one of the following flags to 1.

&SCRGEN-RDF-REC Use for redefining an entire screen record description. The flag invokes the \$screenname-RECORD-RDF macro, which you write.

&SCRGEN-RDF-TAB Use for redefining a screen table. The flag invokes the \$screenname-TABLE-n-RDF macro, which you write.

&SCRGEN-RDF-FLD Use for redefining a screen field. The flag invokes the `$screenname-fieldname-RDF` macro, which you write.

- 2 In the USERMACS macro library, create a macro to redefine your screen record description. Use the following macro naming conventions.

Entire record description	<code>\$screenname-RECORD-RDF</code>
Field in a record description	<code>\$screenname-fieldname-RDF</code>
Table in a record description	<code>\$screenname-TABLE-<i>n</i>-RDF</code>

Where *n* is the table position in sequence with other tables on the screen, counting from top to bottom on the screen.

- Comments:**
- When using the TP-ATTR call in a macro, use the original screen name, not the new (redefined) name.
 - When redefining a screen, include the attribute bytes as part of the redefinition.

Examples: Redefine field PARTNO on screen INVENT. Note that all field names within the screen are prefixed with the screen name.

```
% DEFINE $INVENT-PARTNO-RDF
    &12+05 INVENT-PARTNO-Z99 &36+REDEFINES INVENT-PARTNO
    &40+PIC Z99.
```

Redefine a table definition for a repeat block field named PART-NO-ROW on screen INVENT. The variable `&INVENT-TABLE-1-MAX` contains the number of occurrences for the repeated row block. The rule `$SC-REP-FIELD-HEADER-0` generates the native attribute fields depending on the DC target. The field `INVENT-PART-NO-ROW-EDITED` contains the picture that is used to edit the data for output display.

```
% DEFINE $INVENT-TABLE-1-RDF
    &12+05 FILLER &46+REDEFINES INVENT-TABLE-1.
    &16+10 FILLER &46+OCCURS &INVENT-TABLE-1-MAX.
    $SC-REP-FIELD-HEADER-0("PART-NO-ROW")
    &20+15 INVENT-PART-NO-ROW-EDITED &46+PIC ZZZ9.
% END
```

SCRNLIST

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS, DDS, IMS DC, ISPF Dialog, and ISPF prototyping targets

Description: Enable your program to use multiple screens and generate:

- Procedural code to receive each specified screen
- Screen records in the Working-Storage or Linkage Sections

SCRNLIST automatically generates when you specify multiple I/O screens for a program in the Application Painter. To redefine a single screen, see *Screen Redefinition*.

Syntax CICS

```
[TP-]SCRNLIST screenname1 [... screenname12]
... [MAPSET(mapsetname)]
... [LINKAGE]
... [REDEFINE|NOREDEF]
```

IMS DC

Format 1:

```
[TP-]SCRNLIST screenname1 [.../screenname40]
```

Format 2:

```
[TP-]SCRNLIST screenname1 [... screenname40]
```

ISPF Dialog

Format 1:

```
[TP-]SCRNLIST screenname1[(LK)] [... screenname40[(LK)]]
```

Format 2:

```
[TP-]SCRNLIST screenname1[ ... screenname40]
... [LINKAGE]
```

ISPF prototyping

Any target syntax.

Parameters:	(LK) LINKAGE	Generate every screen record in the Linkage Section instead of in Working-Storage.
	<i>mapsetname</i>	Mapset containing the screen(s) received by the program; must be a literal (maximum 7 characters).
	NOREDEF	Screen records do not redefine each other. Ignored when coded with LINKAGE.
	REDEFINE	Default. Screen records redefine each other.

- Comments:**
- Code the SCRNLIST call to override the generated TP-SCRNLIST, or if coding outside the APS Painters.
 - Code SCRNLIST only once, before NTRY, and do not code a screen name with NTRY.
 - APS generates 88-level flags indicating the received screen.

```
TP-SCRN-RECEIVED          PIC X(08)
      88 TP-screenname-RECEIVED  VALUE 'screenname' .
```

CICS

To generate multiple-map mapsets, access APS Utilities Menu, select the APS Precompiler screen, and then select option 2X, Generate BMS Multiple Map Mapset.

IMS DC

- Label each screen in the APS Screen Painter. A labeled screen contains eight extra bytes (appended to the MID) that contain the screen name. NTRY logic determines which screen named in SCRNLIST was received, moves the data to the appropriate screen record description in Working-Storage, and performs any specified field editing.
- Screens separated with a slash (/) instead of a space redefine each other in Working-Storage, limiting the size of Working-Storage.
- A maximum of 30 screens can be listed in the Application Painter (and thus appear in the automatically-generated TP-SCRNLIST).

- You can write macros to customize SCRNLIST processing for some or all screens in the screen list, using an APS-supplied predefined macro name.. APS executes the macros at locations immediately before and after the following APS-generated paragraphs.

APS Paragraph	Paragraph Function
APS-CHK-SCRN-RECEIVED-PARA	Checks which screen was received
APS- <i>screenname</i> -RECEIVED-PARA	Receives the screen
APS-SCRNLIST-EDIT-PARA	Determines which screen has field edits
APS- <i>screenname</i> -INP-EDIT-PARA	Processes screen field edits

Define the macros using these predefined macro name formats.

```
$TP-PRE-CHK-SCRN-RECEIVED
$TP-POST-CHK-SCRN-RECEIVED
$TP-PRE-screenname-RECEIVED
$TP-POST-screenname-RECEIVED
$TP-PRE-SCRNLIST-EDIT
$TP-POST-SCRNLIST-EDIT
$TP-PRE-screenname-INP-EDIT
$TP-POST-screenname-INP-EDIT
```

Examples: CICS

Generate screen records SCRA and SCRB, that redefine each other, in Working-Storage. Both screens are in mapset SCRASET.

```
SCRNLIST SCRA SCRB MAPSET(SCRASET)
```

Generate a screen record in the Linkage Section for screen SCRA.

```
SCRNLIST SCRA MAPSET(SCRASET) LINKAGE
```

Generate screen records that do not redefine each other.

```
SCRNLIST SCRA SCRB NOREDEF
```

IMS DC

Code logic that receives either SCREENA, SCREENB, or SCREENC, and always sends SCREENC as the output screen:

```
IF TP-SCREENA-RECEIVED
  SCREENC-FIELD-1 = SCREENA-FIELD-X
  SCREENC-FIELD-2 = SCREENA-FIELD-Y
```

```

        SCREENC-FIELD-3 = SCREENA-FIELD-Z
        .
        .
IF TP-SCREENB-RECEIVED
        SCREENC-FIELD-1 = SCREENB-FIELD-XX
        SCREENC-FIELD-2 = SCREENB-FIELD-YY
        SCREENC-FIELD-3 = SCREENB-FIELD-ZZ

```

Code logic where, during a single execution, the program can receive either SCREENA or SCREENB. Let SCREENA and SCREENB redefine each other; keep SCREENC separate (not involved in the redefinition in the SCRNLIST call) in order to prevent any improper overlaying (and thus destruction) of data fields.

```
SCRNLIST SCREENA/SCREENB SCREENC
```

ISPF Dialog

Generate screen records SCRA and SCRB in Working-Storage:

```
SCRNLIST SCRA SCRB
```

Generate screen record SCRA in Working-Storage, and screen record SCRB in the Linkage Section

```
SCRNLIST SCRA SCRB(LK)
```

SD

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Description: Include a sort file description.

Syntax: -KYWD- 12-*----20---*----30---*----40---*----50---*----60
 SD *sortfilename*
 [Applicable COBOL SD clauses]

- Comments:**
- Use one SD keyword per sort file description.
 - Follow each file description with the file record description, using the *01*, *DS*, *REC*, or *++* keywords.

Example:

```

-KYWD- 12-*----20---*----30---*----40---*----50---*----60
IO      INPUT-FILE ASSIGN TO UT-S-INPUT
IO      OUTPUT-FILE ASSIGN TO UT-S-OUTPUT
SPNM    C01 IS TOP-OF-PAGE
FD      INPUT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 0 RECORDS
01      INPUT-RECORD          PIC X(80)
        .
        .
FD      OUTPUT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 0 RECORDS
01      OUTPUT-RECORD        PIC X(80)
        .
        .
SD      SORT-FILE
        RECORD CONTAINS 80 CHARACTERS
        DATA RECORD IS SORT-RECORD
01      SORT-RECORD          PIC X(80)
        .
        .

```

SEARCH

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Use in the same manner as the COBOL SEARCH verb, including conditionals and looping for each WHEN condition.

Syntax: Format 1:

```

SEARCH identifier1 VARYING indexname|identifier2
... [[AT] END]
    statementblock
[WHEN searchcondition1
    statementblock ]
    .
    .
    .
[WHEN searchconditionN
    statementblock ]

```

Format 2:

```
SEARCH ALL identifier [[AT] END]
      statementblock
WHEN searchcondition
      statementblock
```

Example: Use SEARCH to achieve the NEXT SENTENCE concept. If the condition in lines 1040-1050 is true, pass control to line 1110.

```
-LINE- -KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-
001010  PARA   SEARCH-TABLE
001020                SEARCH STOCK-ELEMENT AT END
001030                PERFORM END-ROUTINE
001040                WHEN QUAN-ON-HAND (STOCK-INDEX) NOT <
001050                ... QUAN-NEEDED (STOCK-INDEX)
001060                WHEN QUAN-ON-HAND (STOCK-INDEX) = ZERO
001070                PERFORM NO-STOCK-ROUTINE
001080                WHEN QUAN-ON-HAND (STOCK-INDEX) <
001090                ... QUAN-NEEDED (STOCK-INDEX)
001100                PERFORM DETERMINE-IF-ORDER-ROUTINE
001110                WORK-QUAN = QUAN-ON-HAND (STOCK-INDEX)
```

SEND

Category: Data communication call (see *Data Communication Calls*)

Description: Display screen data for end user response. Additionally:

- Under CICS, generate a CICS SEND MAP command to send screen data to a terminal for user response, as well as a CICS RETURN command to return control to CICS.
- Under IMS DC, send screen messages, in screen or record layout form, to terminals or printers.

Syntax: CICS

```
[TP-]SEND screen[(mapsetname)] [errorpara]
... [TRANSID(name)]
... [NORETURN] [NOERASE]
... [CICSoption [CICSoption] ...]
```

ISPF Dialog

```
[TP-]SEND screen [errorpara]  
... [CONTINUE|NOCONTINUE]
```

IMS DC

```
[TP-]SEND screenname|recordname [errorpara]  
... [lterm]  
... [keyword[+keyword] ...]
```

ISPF Prototyping

Valid syntax is the CICS syntax, the ISPF Dialog syntax, and:

```
[TP-]SEND screenname|recordname [errorpara]  
... [lterm]
```

Parameters:	CONTINUE	Execute the next instruction after the call. See also "Comments" below.
	<i>errorpara</i>	User-defined error routine to perform when an abnormal condition occurs. <i>errorpara</i> is positional; if omitted, code an asterisk (*) in its place.
	<i>keyword</i>	Valid keywords are:
	NOALTRESP	Default. Do not use the alternate response IO PCB to send the response to the terminal.
	ALTRESP	Use the alternate response IO PCB to send the response to the terminal.
	NOCONT	Default. Control returns to the top of the program to process another input message.
	CONT	Execute the next instruction after the call.
	CONTCOND	TP-CONTCOND determines if control passes to the next instruction or returns to the top of the program.

NOEXPRESS	Default. Do not send a message for abnormal program termination.
EXPRESS	Send a message at program termination.
NOENDCONV	Default. Do not blank out TRANCODE in the SPA.
ENDCONV	Blank out TRANCODE in the SPA.
SCREEN	Default. Input is an APS-painted screen. Multisegment screens are not supported.
RECORD	Input is recordname. See also "Comments" below.
NOPURG	Send all messages to the same destination as one multi-segmented message. Default with NOEXPRESS keyword.
PURG	After inserting the message, send it as one single-segmented message. Default with EXPRESS keyword.
<i>lterm</i>	Logical terminal or printer where program sends message; can be a literal (maximum 8 characters) or COBOL data name (maximum 9 characters). Default is device that sends an input message to the program.
<i>mapsetname</i>	Mapset containing the screen(s) the program receives; must be a literal (maximum 7 characters). When not specified, APS generates a default mapset name as per NTRY.
NOCONTINUE	Default. Return control to the top of the program. See also "Comments" below.
NOERASE	Suppress default generation of ERASE.
NORETURN	Suppress default generation of CICS RETURN command (after a generated CICS SEND MAP command).

<i>recordname</i>	User-defined I/O area in Working-Storage. See also "Comments" below.
<i>screenname</i>	Screen name; must be literal (maximum 8 characters).
TRANSID (<i>name</i>)	Transaction code identifying the program where control returns; can be a literal (maximum 4 characters) or COBOL data name (minimum 5 characters).
<i>userparm</i>	Pass linkage data area(s). Code with TP-LINKAGE, which names the 01-level user-defined area in the Linkage Section.

Comments: ISPF Dialog

- With NOCONTINUE, enter your return code checking routine under TP-SCREEN-INVOKED logic. With CONTINUE, enter your return code routine checking under SEND.
- The following APS-provided structure checks the return code after a SEND.

```

APS-TP-SEND-RC          PIC 9(08).
      88 OK-ON-SEND      VALUE 0.
      88 NTF-ON-SEND     VALUE 4.
      88 END-ON-SEND     VALUE 8.
      88 AB-ON-SEND      VALUE 12 16 20

```

IMS DC

- For recordname, move the MOD name to IM-SCREEN-RECORD-MODNAME (an APS-generated field) prior to SEND. Multisegment screens are not supported.
- Use the PURG and NOPURG keywords to control how APS sends messages. When issuing multiple inserts to the same destination before reaching a SYNC point, IMS sends the messages as one multi-segmented message. If you send the message via an Express PCB, APS issues a PURG after the insert and sends it as one single-segmented message.
- When coding RECORD and recordname in conversational programs, code the following statement if recordname differs from your application trancode.

```

SYML    % &recordname-TRANCODE = "trancode"

```

- When sending screens with edited fields, the value of the field variable (the COBOL name assigned during screen painting) moves to a field defined with a display format. The edited fields must contain valid values on output or the screen returns for correction.

To allow end users to send invalid or empty screen values, specify NORETRY with the NTRY keyword and use the generated flags to test for input data validity.

- SEND logically terminates a program when executed; however, the program actually runs as a loop that processes multiple input messages.
- To retain control in the program after sending a screen, code the CONT keyword. You can then send more than one screen or message from a program while processing a single transaction, or send multiple pages of output to the same destination.
- When coding a conversational program in Online Express, APS checks your alternate IO PCB for the following parameters.

```
EXPRESS=YES
MODIFY=YES
ALTRESP=YES
```

If any parameters are missing, a message warns you that, if an error occurs after successful database updates, APS does not send a message to the originating terminal--the program simply terminates and performs a rollback of the updates.

- In a single program execution, the terminal operator can page forward and backward with MFS logical paging commands. To do this:
 - a Ensure that all pages are for the same screen format.
 - b Ensure that all generated APS screens contain a single DPAGE.
 - c Specify operator logical paging on the Screen Generation Parameters screen in the Screen Painter.
 - d Specify that the truncode comes either from a screen field or from PF keys. If PF keys provide all or part of the truncode, define the PF key values on the MFS Function Key screen.

Examples: CICS

Send screen map SCRA.

Program Painter source

```
SEND SCRA
```

Generated source:

```
EXEC CICS SEND MAP( 'SCRA' )
      FROM(AA00-RECORD)
      MAPSET( 'SCRASET' )
      CURSOR
      ERASE
      FREEKB END-EXEC.
EXEC CICS RETURN
      TRANSID( 'SCRA' )
      COMMAREA(TP-COMMAREA)
      LENGTH(169) END-EXEC.
GO TO APS-USER-MAIN-PARA--EXIT.
```

Generate a CICS SEND MAP for map SCRA in mapset SCRASET; return to CICS with transaction code WXYZ. Override the default TRANSID specified in the Screen Painter.

```
SEND SCRA(SCRASET) * TRANSID('WXYZ')
```

Generated source:

```
EXEC CICS SEND MAP( 'SCRA' )
      FROM(AA00-RECORD)
      MAPSET( 'SCRASET' )
      CURSOR
      ERASE
      FREEKB END-EXEC.
EXEC CICS RETURN
      TRANSID( 'WXYZ' )
      COMMAREA(TP-COMMAREA)
      LENGTH(169) END-EXEC.
GO TO APS-USER-MAIN-PARA--EXIT.
```

Generate a CICS SEND MAP for map SCRA.

```
SEND SCRA * NORETURN
```

Generated source:

```
EXEC CICS SEND MAP( 'SCRA' )
      FROM(AA00-RECORD)
```

```
MAPSET( 'SCRASET' )
END-EXEC.
```

DDS and DLG

Display screen SCRA. Determine whether END or RETURN was entered on the screen.

```
NTRY SCRA
  IF TP-PROGRAM-INVOKED
    PERFORM INITIALIZE-SCREEN-FIELDS
  ELSE-IF TP-SCREEN-INVOKED
    IF END-ON-SEND
      ... OR SCRA-FUNCTION = 'E'
      /* USER ENTERED END OR RETURN
      TERM
    ELSE-IF OK-ON-SEND
      PERFORM PROCESS-SCREEN-DATA
    ELSE
      SCRA-SYMSG = 'INVALID OPTION'
  SEND SCRA * NOCONTINUE
```

Display screen SCRA. Determine whether END or RETURN was entered on the screen. Perform return code checking immediately following the SEND.

```
REPEAT
  SEND SCRA * CONTINUE
UNTIL APS-TP-SEND-RC > 0
  PERFORM PROCESS-SCREEN-DATA
TERM
```

IMS DC

Send the single screen defined as output for the program.

```
SEND
```

In a program with multiple output screens, qualify the screens to send.

```
SEND SCRA
```

Send a screen using the EXPRESS PCB.

```
SEND SCRA * * EXPRESS
```

Send a screen using an alternate PCB and return to the next instruction following the call.

```
SEND SCRA * * CONT+ALTRESP
```

SOURCE

Category: Report Writer clause (see *Report Writer Structures* and the *APS User's Guide* chapter *Creating Reports with Report Writer*.)

Compatibility: Batch environments

Description: Map Data Division items to output fields.

Syntax: SOURCE [IS] *dataname* [*iterativeexpression*] [PIC *picclause*]
 [BLANK [WHEN] ZERO]
 [CHANGE INDICATE|GROUP INDICATE]
 [JUSTIFIED|JUST [RIGHT]]
 [DATA-NAME [IS] *fieldname*]

**Keywords/
Parameters:**

BLANK WHEN ZERO	Print spaces when SOURCE <i>dataname</i> is zero. COBOL usage rules apply.
CHANGE INDICATE	Print value of SOURCE <i>dataname</i> whenever it changes. See also "Comments:" below.
<i>dataname</i>	Data item being referenced; can be report mock-up field or a Working-Storage field. See also "Comments" below.
DATA-NAME <i>fieldname</i>	Name a sum accumulator established by a SUM or REFERENCE clause. Do not define <i>fieldname</i> in Working-Storage. At generation, APS inserts <i>fieldname</i> after the level number in the generated report group. DATA-NAME moves the value of the internal SUM accumulator to <i>fieldname</i> . Code the DATA-NAME clause when a SUM UPON clause references DETAIL report group, when the program references a sum accumulator, or when a sum accumulator requires a data name for qualification.

GROUP INDICATE	Identify an item, such as a header, that prints on the first occurrence of its report group after a control break or a page advance. You can use GROUP INDICATE when a DETAIL type defines a printable item; specify GROUP INDICATE for elementary items. If the RED keyword does not contain a PAGE or a CONTROL clause, a GROUP INDICATE item prints the first time its DETAIL line prints after INITIATE processing. See also "Comments" below.
<i>iterative expression</i>	Generate multiple SOURCE statements for suffixed data items or elements of an array. See the APS <i>User's Guide</i> chapter <i>Creating Reports with Report Writer</i> .
JUSTIFIED RIGHT	Right justify the field value. COBOL usage rules apply.
PIC <i>picclause</i>	Specify the format of <i>dataname</i> . If <i>dataname</i> is a report mock-up field instead of a Working-Storage field, the next matching COBOL picture in the report mock-up is the picclause for <i>dataname</i> .

- Comments:**
- SOURCE immediately follows the MOCKUP or OVERPRINT clause.
 - SOURCE designates that the item is described in the program Data Division, is the special Report Writer register LINE-COUNTER or PAGE-COUNTER, or is the internal sum accumulator established by Report Writer.
 - In CONTROL FOOTING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING report groups, SOURCE cannot reference controls or data items referencing controls.
 - To map PAGE-COUNTER when there is more than one report in a program, code SOURCE PAGE-COUNTER OF reportname.
 - In a DETAIL type entry, use CHANGE INDICATE instead of GROUP INDICATE to suppress printing the item after a page break.
 - In a report mock-up, use PIC clauses instead of COBOL masks when formatting dates and times containing / \$ or :. See *Report Mock-Ups*.

Example: -KYWD- 12-*----20---*----30---*----40---*----50---*----60
 01 DETAIL-LINE TYPE IS DETAIL.
 MOCKUP LINE 16
 SOURCE WS-LOCATION-CODE GROUP INDICATE
 SOURCE WS-LAST-COUNT-MONTH PIC 99
 SOURCE WS-LAST-COUNT-DAY PIC 99
 SOURCE WS-LAST-COUNT-YEAR PIC 99
 SOURCE WS-QTY-IN-STOCK JUSTIFIED RIGHT
 SOURCE WS-QTY-ISSUED JUSTIFIED RIGHT
 SOURCE WS-QTY-RECEIVED JUSTIFIED RIGHT
 REFERENCE WS-NO-OF-SALES PIC 9999

Special Registers

Compatibility SQL target

Description: Use selected APS/SQL calls to reference and test special registers as column values. Special registers include:

```
CURRENT DATE
CURRENT TIME
CURRENT TIMESTAMP
CURRENT TIMEZONE
```

Use special registers in DB-DECLARE, DB-OBTAIN, and DB-PROCESS calls to:

- Store and display the special register values.
- Evaluate special register values in WHERE clauses for conditional processing.

Comment: Do not use SQL function names, such as HOUR, TIMESTAMP, as column names.

Examples: Store the value of CURRENT-DATE in a Working-Storage field.

```
DB-OBTAIN REC D2TAB-REC
... PM_PART_NO
... PM_COLOR (WS-COLOR)
... CURRENT DATE (WS-CURR-DATE)
... WHERE PM_PART_SHORT_DESC = 'WIDGET'
... AND PM_COLOR = 'RED'
```

Select rows by comparing column SHIP_DATE with the special register CURRENT DATE.

```
DB-PROCESS REC D2INVEN-REC
... DB-PROCESS-ID D2INV-ID
... IN_PART_NO
... IN_PART_SHORT_DESC
... IN_QTY_ONHAND
... WHERE SHIP_DATE = CURRENT DATE
```

SPNM

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Purpose Create a Special-Names paragraph in the generated program.

Syntax

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SPNM   statement1
      .
      .
      .
SPNM   statement1
```

Example:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SPNM   C01 IS TOP-OF-PAGE
```

SQL

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Compatibility: SQL target

Description: Designate a DB2 table or cursor declaration in the Working-Storage or Linkage Section. Additionally, code native SQL statements and pass them through, without translation, to the precompile process.

Syntax: Format 1, Working-Storage and Linkage Sections:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
SQL    SQLdatastructurestatement
      .
      .
      .
```

Format 2, Procedure Division:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
SQL    nativecall
      . . .
```

- Comments:**
- Use one SQL keyword for each SQL call. APS generates EXEC SQL and END-EXEC statements before the first and last statements, respectively.
 - In the Data Division, the SQL keyword designates a DB2 table or one or more cursor declarations in the Working-Storage or Linkage Section. Use one SQL keyword for each SQL call. APS generates EXEC SQL and END-EXEC statements before the first and last statements, respectively.
 - In the Procedure Division, start every SQL Procedure Division statement with the SQL verb, coded in columns 12 through 72. Begin subsequent lines of the statement with continuation ellipses. All statements pass to the precompiler without translation. SQL statements can be coded anywhere after the PARA, PROC, or NTRY keywords.

Examples: Program Painter code:

```
-KYWD- 12--*--20---*----30----*---40---*---50---*---60---
SQL
      DECLARE DSN8.TDEPT TABLE
      ... (DEPTNO CHAR(3) NOT NULL,
      ... DEPTNAME CHAR(36) NOT NULL,
      ... MGRNO CHAR(3) NOT NULL,
      ... ADMRDEPT CHAR(3) NOT NULL)
REC   WS-DEPARTMENT
      DEPARTMENT-NUM          X3
      DEPARTMENT-NAME        X36
      DEPARTMENT-MGR-NO      X3
      DEPARTMENT-ADMIN       X3
```

NTRY

```

SQL
... SELECT DEPTNO, DEPTNAME,
... MGRNO, ADMRDEPT
... INTO :DEPARTMENT-NUM,
... :DEPARTMENT-NAME,
... :DEPARTMENT-MGR-NO,
... :DEPARTMENT-ADMIN
... FROM DSN8.TDEPT
... WHERE DEPTNO > 0
IF SQLCODE > +0
PERFORM ERROR-DISPLAY

```

Generated code:

```

%   &AP-GEN-VER = 1719
%   &AP-PGM-ID = "TSTSQL"
%   &AP-GEN-DC-TARGET = "ISPF"
%   &AP-TP-ENTRY-KYWD-SEEN = 1
%   &AP-SUBSCHEMA = ""
%   &AP-APPLICATION-ID = "TSTSQL"
%   &AP-GEN-DATE = "861219"
%   &AP-GEN-TIME = "16063323"
IDENTIFICATION DIVISION.
PROGRAM-ID.                TSTSQL.
AUTHOR.                    AP-SYSTEM GENERATED.
DATE-WRITTEN.              861219.
DATE-COMPILED.            &COMPILETIME.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.          &SYSTEM.
OBJECT-COMPUTER.         &SYSTEM.
DATA DIVISION.
WORKING-STORAGE SECTION.
  $TP-WS-MARKER
  EXEC SQL INCLUDE SQLCA
  END-EXEC.
  EXEC SQL
  DECLARE DSN8.TDEPT TABLE           /*
    (DEPTNO CHAR(3) NOT NULL,        /*
    DEPTNAME CHAR(36) NOT NULL,      /*
    MGRNO CHAR(3) NOT NULL,         /*
    ADMRDEPT CHAR(3) NOT NULL)      /*
  END-EXEC.
01 WS-DEPARTMENT.
   05 DEPARTMENT-NUM             PIC X(3).
   05 DEPARTMENT-NAME           PIC X(36).
   05 DEPARTMENT-MGR-NO        PIC X(3).

```

```

      05 DEPARTMENT-ADMIN          PIC X(3).
$TP-COMMAREA
$TP-ENTRY ("", "")
      SQL
          ... SELECT DEPTNO, DEPTNAME,
          ... MGRNO, ADMRDEPT
          ... INTO :DEPARTMENT-NUM,
          ... :DEPARTMENT-NAME,
          ... :DEPARTMENT-MGR-NO,
          ... :DEPARTMENT-ADMIN
          ... FROM DSN8.TDEPT
          ... WHERE DEPTNO > 0
IF SQLCODE > +0
      PERFORM ERROR-DISPLAY

```

STOP RUN

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Return control to the operating system.

Syntax: STOP RUN

- Comments:**
- STOP RUN can appear anywhere in an S-COBOL program.
 - APS generates a STOP RUN at the end of the first paragraph of an S-COBOL program and at the end of the first paragraph of a called program that compiles independently of the main program.

STUB

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Purpose: Include a program stub, or a reusable program module, that is available to any program in the application.

Syntax: -KYWD- 12-*----20---*----30---*----40---*----50---*----60
 STUB stubname

- Comments:**
- During generation, APS inserts the program stub where the keyword STUB appears.
 - To continue programming after a stub, code another keyword.

Example:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
NTRY   MSCR
      IF ENTER-KEY
          PERFORM READ-RTN (MSCR-PART-NBR,
              ... MSCR-NEW-PART-NBR,
              ... MSCR-OLD-PART-NBR,
              ... MSCR-SHORT-DESC, MSCR-UNITS,
              ... MSCR-BASE-PRICE,
              ... MSCR-DIMENSIONS,
              ... MSCR-ERR-MSG)
      ELSE
          MSCR-ERR-MSG = 'INVALID PF KEY ENTERED'
      SEND MSCR
STUB   STUBPGM
PARA   PROCESS-PARA
```

Subselect Clause

Compatibility: SQL target

Description: Create subselect clause by embedding a complete DB-OBTAIN call within a WHERE clause.

Syntax: Embed a DB-OBTAIN call by enclosing it in parentheses. The syntax for the embedded DB-OBTAIN is the same as that for a standard DB-OBTAIN.

- Comments:**
- Create subselect code in a DB-DECLARE, DB-OBTAIN, DB-PROCESS, or DB-STORE call.
 - All standard SQL requirements for a subselect clause apply to the embedded DB-OBTAIN.

- The number and type of columns coded must match the DB-OBTAIN column sequence; both calls must specify columns from different tables.
- When you code a DB-STORE call, make sure that you:
 - Make the number of selected columns for DB-STORE match the number of columns in the result table for the subselect.
 - Do not code the same copylibname-REC in both the DB-STORE and subselect calls.
 - Do not code DB STORE FROM dataname.
 - Do not code alternate values for any columns in DB-STORE.
 - Do not enclose the subselect statement in parentheses, as you do for other DB calls.

Example:

```

DB-OBTAIN REC A.D2TAB-REC
... PM_PART_NO PM_UNIT_BASE_PRICE
... WHERE EXISTS
... (DB-OBTAIN REC D2INVEN-REC
... WHERE IN_PART_NO = A.PM_PART_NO)
DB-DECLARE D2MAST-CURSOR D2TAB-REC
... PM_PART_NO PM_UNIT_BASE_PRICE PM_COLOR
... WHERE PM_PART_NO IN
... (DB-OBTAIN
... REC D2INVEN-REC IN_PART_NO
... WHERE IN_QTY_ONHAND > 100)
... AND PM_UNIT_BASE_PRICE BETWEEN 50 and 100
DB-PROCESS REC D2TAB-REC
... PM_PART_NO PM_UNIT_BASE_PRICE PM_COLOR
... WHERE PM_UNITS <
... (DB-OBTAIN REC D2TAB-REC
... AVG(PM_UNITS))
DB-STORE REC D2INVEN-REC
... IN_PART_NO IN_QTY_ONHAND
... DB-OBTAIN REC D2TAB-REC
... PM_PART_NO PM_PART_SHORT_DESC
... WHERE PM_UNITS > '99'

```

SUM

Category: Report Writer clause (see *Report Writer Structures* and the *APS User's Guide* chapter *Creating Reports with Report Writer*.)

Compatibility: Batch environments

Description: Sum data items, and generate an internal SUM accumulator for a SOURCE or REFERENCE data field to be used in a control footing group.

Syntax: SUM|+ [IS] *dataname* [*iterativeexpression*|*dataname*] ...
 [UPON *detlineidentifier* [*detlineidentifier*] ...]
 [RESET [ON] [FINAL] *controlname*]
 [DATA-NAME [IS] *fieldname*]
 [PICTURE|PIC [IS] *picclause*]

Parameters: *dataname* Data item to reference; can be report mock-up field or a Working-Storage field. Define as a numeric data item. See also "Comments" below.

DATA-NAME *fieldname* Name a sum accumulator established by a SUM or REFERENCE clause. Do not define *fieldname* in Working-Storage. At generation, APS inserts *fieldname* after the level number in the generated report group. DATA-NAME moves the value of the internal SUM accumulator to *fieldname*.

Code DATA-NAME when a SUM UPON clause references DETAIL report group, when the program references a sum accumulator, or when a sum accumulator requires a data name for qualification.

iterative expression Generate multiple SUM statements for numerically suffixed data items or elements of an array. See the *Iterative Expressions* topic in the *APS User's Guide* chapter *Creating Reports with Report Writer*"

<i>PIC picclause</i>	Specify the format of dataname. If dataname is a report mock-up field instead of a Working-Storage field, the next matching COBOL picture in the report mock-up is the picclause for dataname. Picclause determines the size of the internal SUM accumulator.
<i>RESET controlname</i>	Set the SUM accumulator to zero after a control break on controlname. If RESET is not coded, the internal SUM accumulator resets to zero after each control break.
<i>UPON detline-identifier</i>	Sum on the printing of a specific detail line, when there is more than one detail line. Detlineidentifier is the name of DETAIL report group.

- Comments:**
- To set up a SUM accumulator, identify the data field with a SOURCE or REFERENCE on the detail line, and sum it with a SUM clause on the control footing line. APS adds the value of dataname1 to its internal sum accumulator each time the dataname1 data item prints for a detail group.
 - SUM is valid for a CONTROL FOOTING report group only. It follows MOCKUP or OVERPRINT clause.
 - The internal SUM accumulator value prints with the CONTROL FOOTING group and then clears.
 - If the description for a printable data item contains a SUM clause, the internal SUM accumulator serves as a data item, that is, Report Writer moves the internal accumulator value to the data item for printing.
 - APS sums data field values into internal SUM accumulators during GENERATE and TERMINATE processing.

Examples: Each time DEBIT-LINE generates, add SOURCE amount to the DEBIT-LINE and CREDIT-LINE amount accumulators.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
01    DEBIT-LINE TYPE IS DETAIL.
      MOCKUP LINE 1
      SOURCE AMOUNT                PIC 9(5)
01    CREDIT-LINE TYPE IS DETAIL.
      MOCKUP LINE 2
      SOURCE AMOUNT                PIC 9(5)
```

```

01 TYPE IS CONTROL FOOTING WS-PRODUCT-CODE.
   MOCKUP LINE 3
   SUM AMOUNT UPON DEBIT-LINE PIC 9(6)
   SUM AMOUNT UPON CREDIT-LINE PIC 9(6)

```

Print CONTROL FOOTING and RESET the SUM accumulator to zero after the final control break.

```

-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
01 TYPE IS CONTROL FOOTING WS-LOCATION-CODE.
   MOCKUP LINE 15 THRU 21
   SUM WS-QTY-IN-STOCK RESET ON FINAL

```

RESET the WS-LOCAL-SALES field to continue accumulating a sum until the WS-REGION-SALES control break occurs.

```

-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
RED SALES REPORT
   CONTROLS ARE FINAL WS-REGION-SALES
   WS-LOCAL-SALES
01 TYPE IS CONTROL FOOTING WS-LOCAL-SALES.
   MOCKUP LINE 5
   SUM WS-LOCAL-SALES RESET ON
   WS-REGION-SALES

```

Move the value of the NO-OF-SALES accumulator to the TOT-SALES field.

```

-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
01 TYPE IS CONTROL FOOTING WS-LOCATION-CODE.
   MOCKUP LINE 12
   SUM NO-OF-SALES DATA-NAME TOT-SALES

```

Print all values for WS-QTY-IN-STOCK on the detail line in the report and the total of those values in the control footing of the report. Print the total only for WS-NO-OF-SALES in the control footing. When the detail line is prints, add the REFERENCE field value to the SUM accumulator.

```

-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
01 DETAIL-LINE TYPE IS DETAIL.
   MOCKUP LINE 16
   SOURCE WS-QTY-IN-STOCK PIC ZZZ,ZZ9
   REFERENCE WS-NO-OF-SALES PIC 9999
01 TYPE IS CONTROL FOOTING WS-LOCATION-CODE.
   MOCKUP LINES 17 THRU 23
   SUM WS-QTY-IN-STOCK PIC Z,ZZZ,ZZ9
   SUM WS-NO-OF-SALES PIC ZZZ9

```

SUPPRESS (IMS DB Option)

Compatibility: IMS DB target

Description: Suppress the generation of IMS database calls when prototyping under ISPF.

Description: The &IM-SUPPRESS-DB-CALL field prevents DB calls from being generated in your program. During prototyping, this call enables you to code APS/IMS DB calls before you are ready to access the database.

This field resides in the APS CNTL file APSDBDC; set it to YES to suppress DB call generation; the default is NO.

Note: In Online Express, this is handled via the Database Calls field in the Express Params screen in Online Express.

SUPPRESS (Report Writer)

Category: Report Writer statement (see *Report Writer Structures* and the APS *User's Guide* chapter *Creating Reports with Report Writer*.)

Description: Inhibit printing a report group named in a USE BEFORE REPORTING clause.

Syntax: SUPPRESS PRINTING

- Comments:**
- Use SUPPRESS only with USE BEFORE REPORTING.
 - This structure suppresses the following report group functions.
 - Line printing
 - LINE NUMBER and NEXT GROUP processing
 - LINE-COUNTER adjusting

- APS generates the COBOL statement MOVE 1 TO PRINT-SWITCH. When this statement is processed the warning message, NEXT GROUP WILL BE SUPPRESSED, displays.

Example:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
NTRY  OPEN INPUT
      OPEN OUTPUT
      PERFORM MAIN-PARA
DPAR  SUPPRESS-CH-REGION SECTION
      USE BEFORE REPORTING CH-REGION
DPAR  SUPPRESS-CH-REGION-PARA
      IF FIRST-FLG = TRUE
      SUPPRESS PRINTING
```

SUPRA

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Code native SUPRA DBMS procedural statements in the Procedure Division of an S-COBOL program.

Syntax: S-COBOL source input

```
WS
01  [viewname] INCLUDE logicalviewname [(userfieldlist)]
.
.
.
    SUPRA suprastatement
    ...  suprastatement
    ... .
    ... .
    ... .
```

COBOL source output

```
WORKING-STORAGE SECTION.
01  [viewname] INCLUDE logicalviewname [(userfieldlist)]
. . .
    suprastatement
    suprastatement
. . .
```

- Comments:**
- SUPRA statements pass through the APS precompiler unchanged.
 - You must code a native SUPRA INCLUDE statement in the Working-Storage Section.
 - PASS can be substituted for SUPRA to get the same results.

SY* Keywords

Description: Specify in a program the location where the generator places source code.

Syntax:

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYBT  source
SYEN  source
SYDD  source
SYFD  source
SYIO  source
SYLT  source
SYLK  source
SYM1  source
SYM2  source
SYRP  source
SYWS  source
```

Locations in Generated Code

Location	Keyword	Explanation
Top of program	SYM1	At the beginning of the program, before rule libraries that you include at the beginning of the program
	SYM2	After rule libraries that you include at the beginning of the program
Environment Division	SYEN	In the Environment Division, after the Special-Names paragraph
	SYIO	In the Input-Output Section, after rule libraries that you include at the beginning of the Input-Output Section

Location	Keyword	Explanation
Data Division	SYDD	At the beginning of the Data Division
	SYFD	In the File Section, after rule libraries that you include at the beginning of the File Section
	SYWS	In the Working-Storage Section, after rule libraries and data structures that you include in Working-Storage
	SYLT	In the Linkage Section, after rule libraries and data structures that you include at the beginning of Linkage
	SYLK	In the Linkage Section, after source code that you include with the SYLT keyword
	SYRP	In the Report Section, after any rule libraries that you include at the beginning of the Report Section
Procedure Division	SYBT	At the end of the program

- Comments:**
- The effect of a SY* keyword ends with the appearance of another keyword in the KYWD column.
 - The generation process shifts the source to start in column 8.

Examples: Include a rule at the top of the program.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYM1   % INCLUDE USERMACS(MY-RULE)
```

Include a rule at the bottom of the program.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYBT   % INCLUDE USERMACS(MY-RULE)
```

Place a variable assignment statement at the top of the program.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYM1   &TP-USER-LEN = 49
```

Include a copybook in Working-Storage.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
SYWS   % INCLUDE COPYLIB(MY-COPYBOOK)
```

Include a copybook in Linkage.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
SYLK   % INCLUDE COPYLIB(MY-COPYBOOK)
```

System Service Calls

Category: IMS system service calls

Compatibility: IMS DB and IMS DC targets

Description: Ensure successful recovery from error conditions. Monitor and control database services; define regular checkpoints from which you can request a service for the database and check for error conditions; perform error recovery from the last successful checkpoint status check.

Syntax: `IM-CHKP pcbname checkpointID`
`... [length1 dataarea1 [... length7 dataarea7`

```
IM-XRST pcbname
... [length1 area1 [... length7 area7]]
... [checkpointID maxiolength]
IM-CHKP-OSVS pcbname checkpointID
IM-DEQ pcbname deqcharacter
IM-GSCD pcbname
IM-LOG pcbname logcode loglength message
IM-ROLB pcbname [msgarea]
IM-ROLL
IM-STAT-DBAS-FULL pcbname
IM-STAT-DBAS-UNFORMATED pcbname
IM-STAT-DBAS-SUMMARY pcbname
IM-STAT-VBAS-FULL pcbname
IM-STAT-VBAS-UNFORMATED pcbname
IM-STAT-VBAS-SUMMARY pcbname
```

Parameters:	<i>checkpointid</i>	An 8-character COBOL data name or a literal that specifies the ID for this checkpoint
	<i>dataarea</i>	Name of the data area designated in Working-Storage
	<i>deqcharacter</i>	A COBOL data name or single character literal string
	<i>length</i>	Length of data area as defined in Working-Storage
	<i>logcode</i>	A COBOL data name or literal character string containing a code that must be greater than or equal to X'A0' and less than or equal to X'E0'
	<i>loglength</i>	Length of record, excluding the 5-byte header
	<i>maxiolenlength</i>	Length of the largest program I/O area; can be variable or literal; default is the longest path call I/O area, or 0 if no path call exists
	<i>message</i>	A COBOL data name or literal string
	<i>msgarea</i>	Name of area in program where IMS returns the message segment being processed
	<i>pcbname</i>	Database view; can be up to 20 characters; default is IO-PCB

- Comments:**
- These calls generate Working-Storage areas as arguments.
 - For checkpoint/restart functions:
 - These calls neither redefine the IMS facilities nor change the considerations for their use. Consult your appropriate IMS reference manuals for information.
 - Checkpoint and restart are normally not applicable to online programs. This is because the database updates associated with processing an online message represent a single work unit, which executes for only a brief period of time.
 - Batch and BMP programs may require this facility based on the expected duration of execution, number of database updates, and the potential for conflict with other concurrent processes.
 - Checkpoint calls establish both an IMS commit point and a place where the program can be restarted. There are two types of checkpoints.

Symbolic checkpoints can specify up to seven data areas in the program to checkpoint. The program restarts from the last checkpoint call issued, before the abend or from a specific checkpoint named in the restart call IM-XRST. APS restores checkpoint areas to the condition they were when the program abended.

Basic checkpoints do not restart the program; you must provide your own logic. No data areas can be restored.

- Use IM-CHKP for both symbolic and basic checkpoint processing.
- Use IM-XRST for symbolic checkpoint and restart processing.
- IM-XRST is the first IMS call executed by the program. Next, the program should interrogate the IM-XRST-AREA field (generated by this macro). If the field is not equal to spaces, perform restart processing.
- A checkpoint cancels all database positioning. After a checkpoint or restart, the program must reestablish the database positioning with fully qualified DB-OBTAIN calls.
- In batch programs, the program needs the compatibility option available in its PSB (CMPAT=YES).
- Symbolic checkpoints do not support OS/VS files; basic checkpoints do not support OS/VS or GSAM files.

Example: The following code is from an APS batch program using a symbolic checkpoint restart.

```
-KYWD- 12-*-----20----*-----30---*-----40---*-----50---*-----60
WS      CHKPT-WORKAREAS
          CHKPT-ID
              FILLERX4 V'CID1'
              CHKPT-ID-CTR    9(4) V 0
              CHKPT-LIMIT S9(5) V 0 COMP-3
              88 CHKPT-LIMIT-REACHED    V+50
WS      CHECKPOINT-AREA-1
          PREV-PART-NO X8    V LOW-VALUES
NTRY
          IM-XRST IO 8 CHECKPOINT-AREA-1
          IF NOT IM-OK
              PERFORM ERROR-PARA
/*      IF IM-XRST-AREA IS NOT BLANK,
/*      PROGRAM IS BEING RESTARTED
```

```

IF IM-XRST-AREA NOT = SPACES
    MOVE IM-XRST-CHECKPOINT TO CHKPT-ID
    TRUE RESTART
ELSE
/*      PERFORM FIRST CHECK POINT
        PERFORM SYMB-CHKPT-RTN
REPEAT
    PERFORM READ-DB
UNTIL END-ON-REC
    PERFORM PROCESS-DB-REC
/*      INCREMENT COUNTER FOR EACH RECORD READ
        CHKPT-LIMIT = CHKPT-LIMIT + 1
        IF CHKPT-LIMIT-REACHED
            PERFORM SYMB-CHKPT-RTN
PARA  SYMB-CHKPT-RTN
/*      INCREMENT CHKPT-ID CNTR
        CHKPT-ID-CTR = CHKPT-ID-CTR + 1
        IM-CHKP IO CHKPT-ID
        ... 8 CHECKPOINT-AREA-1
        IF NOT IM-OK
            PERFORM ERROR-PARA
        CHKPT-LIMIT = 0

```

The following code:

```

$IM-CHKP ("IO", "MYCHKP", 25, "AREA-1",
% ... 37, "AREA-2")
$IM-CHKP ("IO", "MY-BASIC-CHKP-NAME")
$IM-XRST ("IO", 25, "AREA-1")

```

Generates in Working-Storage:

```

01  IM-CBLTDLI-ARGUMENTS.
    05  IM-CHKP    PIC X(4) VALUE 'CHKP'.
    05  IM-DEQ    PIC X(4) VALUE 'DEQ '.
    05  IM-LOG    PIC X(4) VALUE 'LOG '.
    05  IM-STAT   PIC X(4) VALUE 'STAT'.
    05  IM-XRST   PIC X(4) VALUE 'XRST'.
    05  OSVSCHKP  PIC X(8) VALUE'OSVSCHKP'.
    05  IM-CALL-FUNCTION    PIC X(4).
    05  IM-IO-AREA-LEN PIC S9(9) COMP VALUE +0.
    05  IM-IO-MAXAREA-LEN PIC S9(9) COMP VALUE +0.
    05  IM-LEN-25 PIC S9(9) COMP VALUE +25.
    05  IM-LEN-37 PIC S9(9) COMP VALUE +37.
01  IM-LOG-AREA.
    05  IM-LOG-LEN    PIC S9(4) COMP.
    05  FILLER      PIC S9(4) COMP VALUE +0.
    05  IM-LOG-CODE  PIC X.
    05  IM-LOG-RECORD PIC X(55).

```

```

01 IM-DEQ-CHR    PIC X.
01 IM-XRST-AREA.
    05 IM-XRST-CHECKPOINT PIC X(8).
    05 FILLER      PIC X(4) VALUE SPACES.
01 IM-CHECKPOINT-ID  PIC X(8).
01 IM-STAT-FUNCTION.
    05 FILLER      PIC X(4).
    05 IM-STAT-FORMAT PIC X.
    05 FILLER      PIC X(4).
01 IM-STATISTICS  PIC X(120).

```

Generates in the Procedure Division:

```

MOVE 'MYCHKP'
... TO IM-CHECKPOINT-ID
IF IM-IO-MAXAREA-LEN < IM-IO-AREA-LEN
    MOVE IM-IO-AREA-LEN
... TO IM-IO-MAXAREA-LEN
CALL 'CBLTDLI' USING
... IM-CHKP IO-PCB
... IM-IO-MAXAREA-LEN
... IM-CHECKPOINT-ID
... IM-LEN-25 AREA-1
... IM-LEN-37 AREA-2
MOVE IO-PCB-STATUS
... TO IM-STATUS
...     TP-STATUS
MOVE MY-BASIC-CHKP-NAME
... TO IM-CHECKPOINT-ID
CALL 'CBLTDLI' USING
... IM-CHKP IO-PCB
... IM-CHECKPOINT-ID
MOVE IO-PCB-STATUS
... TO IM-STATUS
...     TP-STATUS
MOVE 'MYOSVSCP'
... TO IM-CHECKPOINT-ID
CALL 'CBLTDLI' USING
... IM-CHKP IO-PCB
... IM-CHECKPOINT-ID
... IM-OSVSCHKP
MOVE IO-PCB-STATUS
... TO IM-STATUS
...     TP-STATUS
MOVE 'A' TO IM-DEQ-CHR
CALL 'CBLTDLI' USING
... IM-DEQ IO-PCB
...IM-DEQ-CHR

```

```

MOVE IO-PCB-STATUS
... TO IM-STATUS
...     TP-STATUS
COMPUTE IM-LOG-LEN = 38 + 5
MOVE LOG-CODE-1 TO IM-LOG-CODE
MOVE LOG-MESSAGE-1
... TO IM-LOG-RECORD
CALL 'CBLTDLI' USING
... IM-LOG IO-PCB
... IO-LOG-AREA
MOVE IO-PCB-STATUS
... TO IM-STATUS
...     TP-STATUS
COMPUTE IM-LOG-LEN = 55 + 5
MOVE LOG-CODE-2 TO IM-LOG-CODE
MOVE LOG-MESSAGE-2
... TO IM-LOG-RECORD
CALL 'CBLTDLI' USING
... IM-LOG IO-PCB
... IO-LOG-AREA
MOVE IO-PCB-STATUS
... TO IM-STATUS
...     TP-STATUS
MOVE 'VBAS'
... TO IM-STAT-FUNCTION /* CLR TAIL
MOVE 'S' TO IM-STAT-FORMAT
CALL 'CBLTDLI' USING
... IM-STAT BE1PARTS-PCB
... IM-STATISTICS
... IM-STAT-FUNCTION
MOVE BE1PARTS-PCB-STATUS
... TO IM-STATUS
MOVE BE1PARTS-PCB
... TO IM-DB-PCB
MOVE SPACES
... TO IM-XRST-AREA
IF IM-IO-AREA-LEN > IM-IO-MAXAREA-LEN
    MOVE IM-IO-AREA-LEN
... TO IM-IO-MAXAREA-LEN
CALL 'CBLTDLI' USING
... IM-XRST IO-PCB
... IM-IO-AREA-LEN
... IM-XRST-AREA
... IM-LEN-25 AREA-1
MOVE IO-PCB-STATUS
... TO IM-STATUS
...     TP-STATUS

```

TERM

Category: Data communication call (see *Data Communication Calls*)

Description: Terminate programs and transaction operations.

Under ISPF Dialog, perform internal program cleanup, terminate the program via GOBACK statement, and return control to the calling program.

Syntax: [TP-]TERM

Comments: CICS

- APS generates a CICS RETURN without a TRANSID or COMMAREA. Program control returns to either the next higher-level linked program or to CICS.
- Code TERM in a linked-to program that returns to the calling program and ensures that TP-COMMAREA data is passed back. When returning to a calling program, TP-COMMAREA moves to DFHCOMMAREA.
- Call does not terminate an active PSB that is passed via LINK.

IMS DC

Use TERM after sending output messages (SEND or MSG-SW) with CONTINUE. Do not use TERM prior to sending at least one response; this causes a user terminal in response mode to remain locked, awaiting a response.

Example: Terminate a program when SCRA-FUNCTION = 'E' or PF3 is pressed.

```
IF SCRA-FUNCTION = 'E' OR PF3
  TERM
```

TERMINATE

Category: Report Writer statement (see *Report Writer Structures* and the *Report Writer* chapter in your *APS User's Guide*)

Compatibility: Batch environments

Description: End report processing. TERMINATE:

- Adds all SUM operands and saves their values
- Saves current values of all CONTROL items
- Produces all CONTROL FOOTING report groups beginning with the minor CONTROL FOOTING and ending with FINAL
- Produces PAGE FOOTING and PAGE HEADING report groups if a page break occurred
- Produces REPORT FOOTING report group
- Resets all CONTROL items to their values when TERMINATE was executed

Syntax: `TERMINATE reportname1 [,reportname2] ...`

Parameter: *reportname* Identify the report. Define reportname in a RED statement in the Report Section of the Data Division.

Comments:

- If GENERATE processing was not executed for a report between its INITIATE and TERMINATE, TERMINATE does not produce any report groups or perform any related processing. If INITIATE processing does not execute, TERMINATE does not execute.
- TERMINATE does not close the report file; to do so, code a COBOL CLOSE statement. TERMINATE every INITIATED report before a CLOSE.

TP-BACKOUT

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS and IMS DC targets

Purpose: ABEND the program.

Syntax: CICS

```
TP-BACKOUT [ABORT[ (name) ] |NOABORT]
```

IMS DC

```
TP-BACKOUT [ABORT|NOABORT]
```

Parameters: CICS

ABORT(<i>name</i>)	Invoke CICS ABEND to terminate task. Name specifies a formatted dump of main storage; can be literal or COBOL data name (maximum 4 characters).
NOABORT	Nonfunctional--allowed for compatibility with IMS.

IMS DC

ABORT	Cancel program and do not reschedule.
NOABORT	Default. Return next input message for processing.

Comments: IMS DC

- Use ABORT if the detected error prevents other messages from processing successfully.
- Use NOABORT if the detected error is specific to the processing message.
- This call invokes the IMS dynamic backout feature, canceling all database updates and messages sent since running the program or since receipt of the most recent input message.

- To send a message and use TP-BACKOUT, use the EXPRESS keyword with either SEND or MSG-SW.
- Coding NOABORT calls an IMS ROLB with the current input message discarded; ABORT calls an IMS ROLL.
- The IMS option of issuing a ROLB with an IOAREA to reread the current input message is not supported by TP-BACKOUT. For further information on ROLL and ROLB, see the appropriate IMS manuals.

Examples: The following three examples execute identically. Each terminates a task abnormally by invoking a CICS ABEND code.

```
TP-BACKOUT
TP-BACKOUT ABORT
TP-BACKOUT NOABORT
```

Specify a recovery option and code name to identify a main storage dump related to the task.

```
TP-BACKOUT ABORT( 'PGM1' )
```

TP-COMMAREA

CICS

Description: TP-COMMAREA generates a Working-Storage record for passing data between programs. This record contains APS information as well as user data.

There is no need to code a TP-COMMAREA call; APS generates it. The value of &TP-USER-LEN--the default is 80--determines the size of TP-USERAREA, the user portion of TP-COMMAREA. You can accept the APS default of a single Commarea field, TP-USERAREA, or redefine TP-USERAREA as multiple Commarea fields.

The following calls pass a Commarea.

<i>SEND</i>	Send a screen to the terminal and terminate the program. CICS saves the data stored in the TP-COMMAREA and makes it available to the next program when APS returns the screen to CICS.
-------------	--

<i>LINK</i>	Link to a subprogram, passing it the TP-COMMAREA address. Define COMMAREA identically in each program.
<i>XCTL</i>	Transfer control to another program, passing a copy of TP-COMMAREA. Define TP-COMMAREA identically in both programs.

You can redefine TP-USERAREA to fit program requirements.

- Code the redefinition in the Data Structure Painter. You must also specify the Commarea redefinition in Application View--code its name in the Data Str field and specify its type as CA (Commarea).
- Code the redefinition in the Program Painter. Two keywords are available to redefine TP-USERAREA--CA, where you code the redefinition in Data Structure Painter syntax, CA05 and CA, where you code it in COBOL syntax.

Optionally, you can code the redefinition in a copybook or rule that you include.

Example: Redefine TP-USERAREA in the Program Painter.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
SYM1  % * SET THE LENGTH OF TP-USERAREA TO
      % * 21 BYTES
      % &TP-USER-LEN = 21
CA     AACA-USERAREA
      CA-DATE                X8
      CA-TIME                X8
      CA-CUST-NUMBER        X5
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
SYM1  % * SET THE LENGTH OF TP-USERAREA TO
      % * 21 BYTES
      % &TP-USER-LEN = 21
CA05  AACA-USERAREA
      10 CA-DATE                X8
      10 CA-TIME                X8
      10 CA-CUST-NUMBER        X5
```

Redefine TP-USERAREA in the Data Structure Painter.

```
AACA-USERAREA
  CA-DATE                X8
  CA-TIME                X8
  CA-CUST-NUMBER        X5
```

Each of the above generates:

```

01 TP-COMMAREA
    .
    .
    .
01 FILLER REDEFINES TP-COMMAREA.
   05 FILLER PIC X(40).
   05 TP-USERAREA PIC X(21).
   05 AACA-USERAREA REDEFINES TP-USERAREA.
      10 CA-DATE PIC X(08).
      10 CA-TIME PIC X(08).
      10 CA-CUST-NUMBER PIC X(05).

```

IMS DC

Description: TP-COMMAREA coordinates the placement of the APS-generated Commarea structure with a user-provided redefinition of that structure. Use it only with conversational IMS programs--nonconversational IMS programs do not use a Commarea.

In IMS, the Commarea is called a SPA (Scratch Pad Area). Its reserved prefix includes an IMS LLZZ field, the input trancode, and the APS invocation mode flag.

There is no need to code a TP-COMMAREA call; APS generates it. The value of &TP-USER-LEN--the default is 0--determines the size of TP-USERAREA, the user portion of TP-COMMAREA. You can accept the APS default of a single Commarea field, TP-USERAREA, or redefine TP-USERAREA as multiple Commarea fields. You must define the Commarea in Working-Storage.

To redefine TP-USERAREA as a group-level data structure, the length must be the same as the value specified for &TP-USER-LEN. A CA01 keyword generates an 05-level REDEFINES TP-USERAREA statement.

To generate a single-field Commarea in Working-Storage called TP-USERAREA, assign a value to the APS variable &TP-USER-LEN. Code the following on one line.

```

-KYWD- 12-*----20---*----30---*----40---*----50---*----60
SYM1   % &TP-USER-LEN = number

```

- SYM1 indicates &TP-USER-LEN is a Customizer variable and places the variable at the top of the program.

- `&TP-USER-LEN = number` specifies the size of TP-USERAREA; it should be large enough to store all fields of data on the screen that the program receives. Caution: An undefined `&TP-USER-LEN` defaults to zero; when prototyping, the default is 2048.

Example:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
SYM1  % &TP-USER-LEN = 100
CA05  MY-REDEF
      10 CA-FLD-A          PIC X(10).
      10 CA-FLD-B          PIC S9(4) COMP.
      10 CA-FLD-C          PIC X(20).
```

Generated source:

```
% &TP-USER-LEN = 100
IDENTIFICATION DIVISION
      .
      .
WORKING-STORAGE SECTION.
$TP-WS-MARKER
$TP-COMMAREA
      05 MY-REDEF REDEFINES TP-USERAREA.
      10 CA-FLD-A          PIC X(10).
      10 CA-FLD-B          PIC S9(4) COMP.
      10 CA-FLD-C          PIC X(20).
```

ISPF Dialog

Description TP-COMMAREA generates a Working-Storage record for passing data between programs. This record contains APS information and user data.

There is no need to code a TP-COMMAREA call; APS generates it. The value of `&TP-USER-LEN` - the default is zero - determines the size of TP-USERAREA, the user portion of TP-COMMAREA. You can accept the APS default of a single Commarea field, TP-USERAREA, or redefine TP-USERAREA as multiple Commarea fields. You must define the Commarea in Working-Storage.

To redefine TP-USERAREA as a group-level data structure, the length must be the same as the value specified for `&TP-USER-LEN`. A CA01 keyword generates an 05-level REDEFINES TP-USERAREA statement.

To generate a single-field Commarea in Working-Storage called TP-USERAREA, assign a value to the APS variable &TP-USER-LEN. Code the following on one line.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYM1   % &TP-USER-LEN = number
```

- SYM1 indicates &TP-USER-LEN is a Customizer variable and places the variable at the top of the program.
- &TP-USER-LEN = *number* specifies the size of TP-USERAREA; it should be large enough to store all fields of data on the screen that the program receives. The default for *number* is zero.

To receive TP-COMMAREA as a passed data area from a calling program, code this assignment statement before NTRY.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
      % &DLG-COMMAREA-IN-LINKAGE = "YES"
```

The results are:

- APS generates the following in the Linkage Section.


```
01 DLG-LINKAGE-COMMAREA PIC X(&TP-USER-LEN).
```
- APS generates the Procedure Division statement with


```
... USING DLG-LINKAGE-COMMAREA
```
- Generated logic maps the data between TP-COMMAREA and DLG-LINKAGE-COMMAREA at the following points.
 - Upon program invocation, from DLG-LINKAGE-COMMAREA to TP-COMMAREA
 - Before execution of LINK or XCTL, from TP-COMMAREA to DLG-LINKAGE-COMMAREA
 - After execution of LINK or XCTL, from DLG-LINKAGE-COMMAREA to TP-COMMAREA
 - Before program termination, from TP-COMMAREA to DLG-LINKAGE-COMMAREA

Example: Redefine TP-USERAREA.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYM1   % &TP-USER-LEN = 49
CA05   PGM-USERAREA
```

```

10 CA-EMPLOYEE-NAME    PIC X(20).
10 CA-EMPLOYEE-TITLE   PIC X(20).
10 CA-EMPLOYEE-SSN     PIC X(09).

```

Generated source:

```

01 TP-COMMAREA.
05 TP-USERAREA          PIC (X49).
05 PGM-USERAREA REDEFINES TP-USERAREA.
10 CA-EMPLOYEE-NAME     PIC X(20).
10 CA-EMPLOYEE-TITLE   PIC X(20).
10 CA-EMPLOYEE-SSN     PIC X(09).

```

TP-LINKAGE

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS, IMS DC, and ISPF Dialog

Description: Handle addressability of Linkage Section data records in a called program.

Syntax: TP-LINKAGE *linkdataname*[/*copybookname*|*macrofilename*]
 ... [*linkdataname*[/*copybookname*|*macrofilename*] ...]

Parameters:

<i>/copybookname</i>	Name of copybook that you INCLUDE or COPY prior to this call. See also "Comments" below.
<i>linkdataname</i>	01-level Linkage Section data area identical to the linkdataname in the associated call.
<i>/macrofilename</i>	Name of macro file that you INCLUDE or COPY prior to this call. See also "Comments" below.

Comments:

- List data items in the order that they appear in the Linkage Section, copybook, or macro file.
- Make sure that each linkdataname you specify is defined in the Linkage Section. Choose one of the following methods.
 - Define them yourself.

- Include (using TP-LINKAGE) a copybook or macro file that contains the linkdatanames.
- To generate a COBOL COPY statement, instead of a % INCLUDE statement, code &TP-COPY-LINKAGE = 1 prior to the TP-LINKAGE call.
- In Online Express, list the records in the order they pass from the calling program.

CICS

- Use TP-LINKAGE with calls that use the SET option.
- Coding TP-LINKAGE with the SYLT keyword ensures that the APS-generated DFHCOMMAREA is the first record in the Linkage Section. This prevents the CICS translator from generating an additional one.
- One BLL cell is generated per 4,096 bytes. Linkage records longer than 4,096 bytes require additional BLL cells to establish addressability to the entire record.
- To generate additional BLL cells, code dummy records after the record that is greater than 4,096 bytes. A dummy record name must be prefixed with DUMMY and its copybook name must be DUMMY.
- BLL cells are not used with COBOL/2.

IMS DC

- Code the call on the first line of the Linkage Section, and in conjunction with NTRY or PROC to generate parameters for the PROCEDURE DIVISION USING statement.
- Do not define PCBs with TP-LINKAGE; the subschema defines them.

ISPF Dialog

- APS provides this call for compatibility with APS/CICS and APS/IMS-- it is not required to handle addressability of Linkage Section records.
- Make the linkdataname a PARM area and the only area defined in the Linkage Section, if the PROGRAM CONTROL TRANSFER option is SELECT.

Examples: Define records LINK-REC-1 and LINK-REC-2. Include definitions from USERMACS macro members LINKR1 and LINKR2.

```
TP-LINKAGE LINK-REC-1/LINKR1 LINK-REC-2/LINKR2
```

Define records LINK-REC-1 and LINK-REC-2 and copy their definitions from COPYLIB members LINKR1 and LINKR2.

```
% &TP-COPY-LINKAGE = 1
TP-LINKAGE LINK-REC-1/LINKR1 LINK-REC-2/LINKR2
```

Define records LINK-REC-1 and LINK-REC-2, coded in the Linkage Section.

```
TP-LINKAGE LINK-REC-1 LINK-REC-2
```

Define records LINK-REC-1 and LINK-REC-2. LINK-REC-1 has been coded in the Linkage Section; LINK-REC-2 is copied from COPYLIB member LINKR2.

```
% &TP-COPY-LINKAGE = 1
TP-LINKAGE LINK-REC-1 LINK-REC-2/LINKR2
LINK-REC-1      PIC X(100).
```

CICS

Linkage record containing 10,000 bytes.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
SYLK   TP-LINKAGE LINK-REC
      ... DUMMY-LINK-REC-BLL1/DUMMY
      ... DUMMY-LINK-REC-BLL2/DUMMY
LK01   LINK-REC.
      05 LINK-REC-1-4096      PIC X(4096).
      05 LINK-REC-2-8192      PIC X(4096).
      05 LINK-REC-3-10000     PIC X(1808).
OPT    PROG
NTRY

CIC-GETMAIN
... SET(LINK-REC)
... LENGTH(10000)
/* PROGRAM HAS ADDRESSABILITY TO
/* THE FIRST 4,096 BYTES.
ADD 4096 LINK-REC--P
... GIVING DUMMY-LINK-REC-BLL1--P
... GIVING DUMMY-LINK-REC-BLL2--P
/* PROGRAM HAS ADDRESSABILITY TO
/* ALL 10,000 BYTES.
```

ISPF Dialog

Define a PARM area for use by TP-LINK and TP-ENTRY. The PROGRAM CONTROL TRANSFER option has been set to SELECT. PGM-PARM-DATA is coded inline. Note: You can place the PGM-PARM-DATA area in Linkage without the use of TP-LINKAGE.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYLK   TP-LINKAGE PGM-PARM-DATA
LK01   PGM-PARM-DATA.
        05 PGM-PARM-DATA-LEN   PIC S9(04) COMP.
        05 PGM-PARM-EMPNBR    PIC X(05).
        05 FILLER              PIC X(95).
```

TP-NULL

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS, IMS DC, and ISPF Dialog targets

Description: Move LOW-VALUES to all fields in a specified screen record.

Syntax: TP-|SC-NULL *screenname*

Parameter: *screenname* Screen name; value must be literal.

Comment: This call does not alter the field attributes.

Example: Move LOW-VALUES to all fields on screen SCRA.

```
TP-NULL SCRA
```

TP-PERFORM

Category: Data communication call (see *Data Communication Calls*)

Description: Perform a paragraph, with or without passing arguments.

Syntax: Format 1, perform a paragraph:

```
TP-PERFORM paragraphname
```

Format 2, perform a paragraph and pass arguments:

```
TP-PERFORM paragraphname actualargument1[[[,]
... actualargument2[,] ... actualargument8]
.
.
.
paragraphname ([+|-]formalargument1[[[,]
... [+|-]formalargument2[,] ... [+|-]formalargument8])
```

Parameters: +/-

A plus (+) or minus (-) sign preceding a *formalargument* passes values between actual arguments and formal arguments, as follows:

- With a plus sign (+), PERFORM passes the *actualargument* value to *formalargument* after the paragraph executes. The program does not return the *formalargument* value to *actualargument*.
- With a minus sign (-), PERFORM passes the *formalargument* value to the *actualargument* after the paragraph executes.
- If there is no prefix, the PERFORM passes the *actualargument* to its corresponding *formalargument*. After the paragraph executes, PERFORM passes the *formalargument* back to its corresponding *actualargument*.

<i>actualargument</i>	Values you send or receive from a <i>formalargument</i> in the paragraph; can be literals, identifiers, arithmetic expressions, or index names.
<i>formalargument</i>	Values you receive from or send to an <i>actualargument</i> in the PERFORM statement.
<i>paragraphname</i>	Name of paragraph to perform.

- Comments:**
- When continuing a list of arguments onto one or more other lines, do not break an argument.
 - The number of actual arguments must equal the number of formal argument names and be in the same order.

TRUE/FALSE

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Establish flags, set them to true and false, and then test them.

Syntax: TRUE|FALSE *dataname1* [*dataname2* [... *datanameN*]]

- Comments:**
- For each *dataname* coded, S-COBOL provides an 02-level conditional variable and an associated 88-level condition name entry in an 01 GENERATED-FLAGS area of Working-Storage. APS adds a --FLG suffix to *dataname* to create the 02-level entry, and *dataname* itself becomes the 88-level entry. APS always assigns VALUE 'T' to the 88-level condition name, never VALUE 'F'. For example:

The following code in the Procedure Division:

```

      .
      .
      .
TRUE  THERE-ARE-NO-ERRORS
FALSE WITHIN-RANGE
      .
      .
      .

```

Generates:

```

WORKING-STORAGE SECTION.
.
.
.
02  THERE-ARE-NO-ERRORS--FLG    PIC X.
   88  THERE-ARE-NO-ERRORS      VALUE 'T'.
02  WITHIN-RANGE--FLG          PIC X.
   88  WITHIN-RANGE            VALUE 'T'.

```

- If, for debugging purposes, you want to display a generated flag, add the --FLG suffix to dataname in the DISPLAY statement. For example:

```

TRUE THERE-ARE-NO-ERRORS generates
MOVE TRUE TO THERE-ARE-NO-ERRORS--FLG.

```

- If only true statements, such as TRUE WITHIN-RANGE, or only false statements, such as FALSE WITHIN-RANGE, are coded for dataname, a warning message indicates a logic error.
- S-COBOL minimizes the code required to test the flags established and set by the TRUE/FALSE verbs.
 - Code WHILE THERE-ARE-NO-RECORDS instead of WHILE THERE-ARE-NO-RECORDS--FLG = TRUE.
 - Code IF NOT WITHIN-RANGE instead of IF WITHIN-RANGE--FLG = FALSE.

Example: Set two undefined flags, THERE-ARE-NO-ERRORS and WITHIN-RANGE to TRUE (line 3030). Define these flags in Working-Storage (88-level condition name flags), because this is the first TRUE or FALSE statement for either flag in this program. If the condition in line 3060 is true, set the THERE-ARE-NO-ERRORS flag to FALSE. If either condition in line 3110 is true, set the WITHIN-RANGE flag to FALSE. As long as the value of the THERE-ARE-NO-ERRORS flag is TRUE, perform the loop. Inside the loop test the other flag, WITHIN-RANGE. If the value is TRUE, execute the subordinate statement block.

```

-LINE-  -KYWD-  12-*----20---*----30---*----40---*----50---*
.
003030          TRUE THERE-ARE-NO-ERRORS
003031          ... WITHIN-RANGE
.
003060          IF CODE-IN NOT NUMERIC
003070          FALSE THERE-ARE-NO-ERRORS

```

```

.
003110      IF COUNT > 372 OR COUNT < 50
003120      FALSE WITHIN-RANGE
.
003150      WHILE THERE-ARE-NO-ERRORS
003160      IF WITHIN-RANGE

```

TRUE, FALSE, ALWAYS, NEVER

Category: S-COBOL flag (see *S-COBOL Structures*)

Description: TRUE, FALSE, ALWAYS, NEVER are data names whose meanings are reserved and automatically provided by S-COBOL.

Syntax: Automatically generated in Working-Storage

```

01  GENERATED-FLAGS.
    02  TRUX          PIC X  VALUE 'T'.
        88  ALWAYS   VALUE 'T'.
        88  NEVER    VALUE 'F'.
    02  FALSX        PIC X  VALUE 'F'.

```

Comment: Because TRUE and FALSE are reserved words on some systems, S-COBOL changes every occurrence of TRUE and FALSE to TRUX and FALSX, and generates the flags accordingly.

TYPE

Category: Report Writer statement (see *Report Writer Structures* and the *User's Guide* chapter *Create Reports with Report Writer*)

Compatibility: Batch environments

Description: Identify a report group, such as a header line, detail line, or footer line.

Syntax: Format 1, page header:

```

TYPE [IS] PAGE HEADING|PH

      [LINE [NUMBER IS] number      ] [.]
      PLUS number

```

Format 2, page footer:

```

TYPE [IS] PAGE FOOTING|PF
      [LINE [NUMBER IS] number]

      [NEXT GROUP [IS] number      ] [.]
      PLUS number

```

Format 3, report header:

```

TYPE [IS] REPORT HEADING|RH

      [LINE [NUMBER IS] number      ]
      PLUS number

      number
      [NEXT GROUP [IS] PLUS number] [.]
      NEXT PAGE

```

Format 4, report footer:

```

TYPE [IS] REPORT FOOTING|RF

      number
      [LINE [NUMBER IS] PLUS number] [.]
      NEXT PAGE

```

Format 5, control headers and footers:

```

TYPE [IS] CONTROL HEADING|CH [FINAL] controldataname
      CONTROL FOOTING|CF
      number

      [LINE [NUMBER IS] PLUS number]
      NEXT PAGE
      number

      [NEXT GROUP [IS] PLUS number] [.]
      NEXT PAGE

```

Format 6, detail lines:

detaildataname TYPE [IS] DE[TAIL]

number
[LINE [NUMBER IS] PLUS *number*]
NEXT PAGE

number
[NEXT GROUP [IS] PLUS *number*] [.]
NEXT PAGE

**Keywords/
Parameters:**

CONTROL FOOTING| CF Print group totals immediately following the detail lines each time a control group ends, that is, when a control break occurs.

CONTROL HEADING| CH Print heading line(s) before each detail group, that is, when a control break occurs.

controldataname Designate control data name. Unless FINAL, code *controldataname* in the corresponding RED keyword CONTROL clause.

DETAIL|DE Specify the body group, that is, lines containing data items of a report. See also "Comments" below.

detaildataname Name of detail line.

FINAL Specify the highest, most inclusive, control group. Implicit; does not have to be coded in order to be used in a CONTROL FOOTING.

LINE *number* Designate the line number where the current header, footer, or detail line prints. *Number* (maximum 3 digits) must be within the defined page limits. Specify the RED keyword PAGE LIMIT clause.

LINE NEXT PAGE Print the current header, footer, or detail line on a new page.

LINE PLUS <i>number</i>	<p>Designate the line where the current header, footer, or detail line prints, and optionally insert blank lines. <i>Number</i> (maximum 3 digits) must be within the defined page limits.</p> <p>PLUS increments the line number by <i>number</i>, causing blank lines. A simpler way to print blank lines, however, is to include them in your mock-up.</p>
NEXT GROUP <i>number</i>	<p>Designate the line number where the next TYPE entity prints (for example, a detail line, header, or footer). <i>Number</i> (maximum 3 digits) must be within the defined page limits. Specify the RED keyword PAGE LIMIT clause. See also "Comments" below.</p>
NEXT GROUP PLUS <i>number</i>	<p>Designate the line where the next TYPE entity prints, and optionally insert blank lines (for example, a detail line, header, or footer). <i>Number</i> (maximum three digits) must be within the defined page limits.</p> <p>PLUS increments the line number <i>number</i>, causing blank lines. A simpler way to print blank lines, however, is to include them in your mock-up. See also "Comments" below.</p>
NEXT GROUP NEXT PAGE	<p>Print the next TYPE entity on a new page (for example, a detail line, header, or footer). Do not code NEXT PAGE with PAGE FOOTING.</p>
PAGE HEADING PH	<p>First line(s) on each page. APS processes PAGE HEADING as the first report group on each page, unless a REPORT HEADING, that is not on a page by itself, precedes it. APS ignores PAGE HEADING on a page that contains only a REPORT HEADING or REPORT FOOTING. See also "Comments" below.</p>
PAGE FOOTING PF	<p>Last line(s) on each page. APS processes PAGE FOOTING as the last report group on each page of a report, unless a REPORT FOOTING, that is not on a page by itself, follows it. APS ignores PAGE FOOTING on a page that contains only a REPORT HEADING or REPORT FOOTING. See also "Comments" below.</p>

REPORT FOOTING|RF Last line(s) of the report. The TERMINATE statement processes REPORT FOOTING as the last report group.

REPORT HEADING|RH First line(s) of the report. It is the first report group and processes once per report.

- Comments:**
- Enter at least one TYPE statement per report.
 - Code a TYPE statement for each report group in the report, that is, each type of report line.
 - Code at least one detail line for each report, including a summary report. Without a detail line, SUM accumulators are not summed.
 - APS ignores NEXT GROUP with CONTROL FOOTING, unless CONTROL FOOTING is at the highest level for a control break.
 - Code LINE NUMBER and NEXT GROUP clauses on separate lines.
 - Specify the PAGE HEADING and PAGE FOOTING report groups only if the RED statement has a PAGE LIMIT clause. Specify the CONTROL HEADING or CONTROL FOOTING in the RED CONTROL clause. FINAL is implicit.

Examples:

```

01      TYPE IS REPORT HEADING
          NEXT GROUP IS NEXT PAGE.
01      TYPE IS PAGE HEADING.
          SOURCE IS PAGE-COUNTER          PIC ZZZ9
01      PART-DETAIL TYPE IS DETAIL.
          MOCKUP LINES 5 THRU 6
          SOURCE IS PM-PART-NO           PIC XXXXXXXX
          SOURCE IS PD-LONG-DESC        PIC X(50)
          SOURCE IS PM-UNIT-BASE-PRICE  PIC $$$,$$9.99
01      TYPE IS CONTROL FOOTING.
          MOCKUP LINES 7
          SUM PM-UNIT-BASE-PRICE        PIC $$$,$$9.99

```

Print a report heading on line 20.

```

-KYWD- 12-*----20---*----30---*----40---*----50---*----60
01      TYPE IS REPORT HEADING LINE 20.

```

Insert blank lines in the report by incrementing the line number by two before printing the footer.

```

-KYWD- 12-*----20---*----30---*----40---*----50---*----60
01      TYPE IS REPORT FOOTING LINE PLUS 2.

```

Print two blank lines between groups. Indentation causes continuation when coding NEXT GROUP PLUS 2.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
01     TYPE IS CONTROL FOOTING WS-LOCATION-CODE
        NEXT GROUP PLUS 2.
```

Increment the page counter after printing a line.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
01     TYPE IS REPORT HEADING
        NEXT GROUP NEXT PAGE.
```

UNION

Category: Database access clause

Compatibility: SQL target

Description: Unite a DB-DECLARE or DB-PROCESS call with one or more DB-OBTAIN calls via the UNION keyword, which collects similar columns from two or more tables into one new table. The DB-OBTAIN calls in a union can select rows from one or many tables; the union results in a single table containing the rows selected by each call.

Syntax: With DB-DECLARE:

```
DB-DECLARE cursorname [correlname1.]copylibname-REC
.
.
... UNION [ALL]
DB-OBTAIN REC copylibname-REC
.
.
.
... [ORDER
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]
```

With DB-PROCESS

```

DB-PROCESS REC [correlname1.] copylibname-REC
      .
      .
      .
... UNION [ALL]
DB-OBTAIN REC copylibname-REC
      .
      .
      .
... [ORDER
... column1 [ASC|DESC] [...columnN [ASC|DESC]]]

```

Parameters: See the applicable database call for parameter descriptions.

- Comments:**
- To create a union of tables, code all keywords and parameters for each DB-DECLARE, DB-OBTAIN, and DB-PROCESS call just as you do for a single call.
 - When coding a UNION, the columns selected in the DB-PROCESS or DB-DECLARE statements in the UNION determine the host variables the FETCH uses. Specify return-fields only in the DB-DECLARE or DB-PROCESS statements of the UNION. See the last example below.
 - Because a UNION places all rows in a single table, code INTO only in the last DB-OBTAIN call. However, if a DB-OBTAIN call joins tables, do not code INTO, which puts the results from only one table's host-area into the alternate area. A joined table has only part of its contents placed into the alternate area named by INTO.
 - The column selection list for DB-DECLARE, DB-PROCESS, and each DB-OBTAIN must adhere to standard SQL rules for UNION.
 - To identify the source call that produces each row in a UNION, include numeric and character literals in your column specifications for each call. Each row then includes a column that identifies the specific DB-DECLARE, DB-OBTAIN, or DB-PROCESS call that retrieved it.
 - Although you can code a literal for a column in any APS/SQL call, its primary use is for UNIONS.
 - UNION ALL includes all rows selected from the source tables in the new table. To eliminate duplicate rows from the new table, do not code ALL.

- APS/SQL statements adhere to all other SQL requirements for tables created through a union.
- UNION combines tables with the same number of columns. The rows of each table must contain column sequences that match in data type and length.
- To sort rows in a table, code the ORDER clause, as follows.
 - Code ORDER only once, as the final statement in the last DB-OBTAIN call. The ORDER clause applies to the result table produced by UNION.
 - Specify sort columns by their position in the selection list (for example, 1 or 2), rather than by name.

Examples: Unite DB-DECLARE with one DB-OBTAIN; eliminate duplicate rows; sort the combined table in ascending order by PM_PART_NO, then in descending order by PM_UNIT_BASE_PRICE and PM_UNITS. Note that the column literals STMT1 and STMT2 identify which call retrieves each row.

```
DB-DECLARE D2MAST-CURSOR D2TAB-REC
... DISTINCT
... PM_PART_NO PM_UNIT_BASE_PRICE PM_UNITS 'STMT1'
... WHERE PM_PART_SHORT_DESC = :WS-PART-SHORT-DESC
... AND PM_UNIT_BASE_PRICE BETWEEN 50 AND 150
... UNION
DB-OBTAIN REC D2TAB-REC
... PM_PART_NO PM_UNIT_BASE_PRICE PM_UNITS 'STMT2'
... WHERE PM_PART_SHORT_DESC = :WS-PART-SHORT-DESC
... AND PM_UNIT_BASE_PRICE > 300
... AND PM_UNITS > 1000
... ORDER 1,2 DESC, 3 DESC
```

Unite DB-PROCESS with one DB-OBTAIN call. Sort the combined table in ascending order by PM_PART_NO, then in descending order by PM_UNIT_BASE_PRICE and PM_UNITS.

```
DB-PROCESS REC D2TAB-REC
... DB-PROCESS-ID D2UNION-ID
... DISTINCT
... PM_PART_NO PM_UNIT_BASE_PRICE PM_UNITS
... WHERE PM_PART_SHORT_DESC = :WS-PART-SHORT-DESC
... AND PM_UNIT_BASE_PRICE > 50
... AND PM_UNIT_BASE_PRICE < 150
... DB-LOOP-MAX=500
... UNION
```

```

DB-OBTAIN REC D2TAB-REC
... PM_PART_NO PM_UNIT_BASE_PRICE PM_UNITS
... WHERE PM_PART_SHORT_DESC = :WS-PART-SHORT-DESC
... AND PM_UNIT_BASE_PRICE > 300
... AND PM_UNITS > 1000
... ORDER 1,2

```

Because return-fields are not specified for JOB_NAME and PROC_NAME, the host variables of those same names are used. In the case of the literal P, no host variable exists so a return-field must be specified. Specify return-fields only in the DB-DECLARE or DB-PROCESS statements.

```

DB-PROCESS
... REC A.HTJCLD-REC
... DB-PROCESS-ID SHARED-ID
... JOB_NAME
... PROC_NAME
... REC B.HTJOBR-REC
... 'P' (WS-HOLD-ACTION)
... WHERE B.RESOURCE = :WS-RESOURCE AND
... B.JOB_NAME = :WS-JOB-NAME
... UNION ALL
DB-OBTAIN REC A.HTJCLD-REC
... DISTINCT JOB_NAME
... PROC_NAME
... REC B.HTJSTEP-REC
... 'A'
... WHERE B.JOB_NAME = :WS-JOB-NAME
... ORDER 01 03 02

```

UNTIL/WHILE

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Form a loop with a test at the beginning that allows a subordinate statement block to execute repeatedly, either until or while a single or compound condition is satisfied.

Syntax: Format 1:

```

REPEAT
    .
    .
    .
UNTIL/WHILE condition1 [AND/OR condition2
... [... AND/OR conditionN]]
    statementblock

```

Format 2:

```

REPEAT
    statementblock1
UNTIL condition
    statementblock2

```

Format 3:

```

REPEAT
    statementblock1
UNTIL condition
    statementblock2
statementblock3

```

- Comments:**
- Use these statements with REPEAT to test a condition at the middle or end of a loop.
 - In the Format 2, APS tests the condition after the REPEAT *statementblock1* executes. Be careful using this format for reading records--it can read the last record twice.
 - In the Format 3, APS tests the condition after the REPEAT *statementblock1* executes, but before the UNTIL *statementblock2* executes. When the UNTIL condition is true, the UNTIL *statementblock2* does not execute.

Examples: Find the first X in a string of characters.

```

-KYWD- 12-*-----20----*-----30----*-----40----*-----50----*-----60
PARA   FIND-X
        A-SUB = 1
        UNTIL A-SUB > 100
        ... OR CHAR (A-SUB) = "X"
        A-SUB = A-SUB + 1
        IF A-SUB <= 100
            CHARACTER-COUNT = A-SUB
        ELSE

```

.
.
.

Use WHILE to achieve the same results.

```
WHILE A-SUB <= 100
... AND CHAR (A-SUB) NOT = 'X'
```

USE BEFORE REPORTING

Category: Report Writer clause (see *Report Writer Structures* and the *APS User's Guide* chapter *Create Reports with Report Writer*)

Compatibility: Batch environments

Description: Code declarative procedures to modify any heading or footing report group before it prints. You can specify special processing requirements, including calculations in addition to those specified in the SUM clause, and edits to a report line.

Syntax: USE BEFORE REPORTING *identifier*

Parameters: *identifier* Designate a HEADING or FOOTING report group. *Identifier* cannot appear in more than one USE BEFORE REPORTING.

- Comments:**
- USE BEFORE REPORTING identifies declarative statements that execute before HEADING or FOOTING report groups print. You name the report group in the Report Section of the Data Division.
 - A USE statement follows a section header in a declaratives section and ends with a period and space. The section can consist of paragraphs that define procedures.
 - A USE BEFORE REPORTING statement cannot alter the value of any control data item.
 - The INITIATE, GENERATE, or TERMINATE clauses cannot appear in a paragraph within a USE BEFORE REPORTING.

- Report Writer ensures that the designated statements execute just before the specified report group.
- Declarative statements cannot reference non-declarative statements; conversely, non-declarative statements cannot reference a statement name appearing in the declarative portion. The COBOL PERFORM statement is the only exception of a non-declarative that refers to a declarative statement.

Example:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
NTRY  OPEN INPUT
      OPEN OUTPUT
      PERFORM MAIN-PARA
DPAR  SUPPRESS-CH-REGION SECTION
      USE BEFORE REPORTING CH-REGION
DPAR  SUPPRESS-CH-REGION-PARA
      IF FIRST-FLG = TRUE
      SUPPRESS PRINTING
```

User Help

Category: User application Help

Description: Create the help source file.

Procedure To create the help source file, follow these steps,

- 1 From the APS Main Menu, enter option **2** in the **Command** field. Then enter option **6** in the **Command** field. APS displays the User Help Facility screen.
- 2 From the User Help Facility screen, enter **1** in the **Command** field. APS displays the User Help Source Utility screen.
- 3 Select a utility to create your help source file. If you select , Applications, APS displays the Applications Utility screen. If you select Data Elements, APS displays the Data Elements Utility screen. If you select Screens, APS displays the Screens Utility screen.

4 Complete the fields for the utility selected.

Field	Screen	Description
Context Name	Data Elements	Enter the context name associated with the field to display all the fields with that context. Leave this field blank to display the fields with no context. Enter all to display all the fields with their contexts.
Context List	Data Elements	Display a context list.
Application Name	Applications	Enter the user application name, or leave this field blank and press Enter to display a selection list. Select a name from the selection list by entering s next to it.
Field Name	Data Elements	Type a field name or leave this field blank to display a selection list.
Screen Name	Screens	Type a screen name, or leave this field blank to display a selection list.
Edit Business Name	All	Select to assign a business name a descriptive name that easily identifies the user application and its components). If you do not assign a business name, it defaults to the user application name.
Edit text	All	Select to create help text.
Include Screens	Applications	Select to create screen help.
Include Fields	Applications	Select to create field help.
Local Fields	Applications and Screens	Select to create field help.

Field	Screen	Description
Create Values	All	Select to create field value help.
Help Source File Name	All	Help source file name, TMP.APSEXT. If this file already exists, it is overlaid. Note: Do not use an extension with this filename.

Examples:

```

COMMAND ==>
Specify the items to be included in the extract, then press ENTER

Application name      ==>      (Blank for entity list)
Edit business name   ==> NO   (Yes or No)
Edit text            ==> NO   (Yes or No)

Include screens      ==> NO   (Yes or No)
IF YES:
  Edit business name ==> NO   (Yes or No)
  Edit text          ==> NO   (Yes or No)

Local fields         ==> NO   (Yes or No)
IF YES:
  Edit business name ==> NO   (Yes or No)
  Edit text          ==> NO   (Yes or No)

Create values        ==> NO   (Yes or No)
IF YES:
  Edit text          ==> NO   (Yes or No)

Help source file name:
==> C:\TMP\APSEXT

```

```

COMMAND ==>
Specify the items to be included in the extract, then press ENTER

Context              ==>      (Name, blank, or ALL)
Context list         ==> NO   (Yes or No)

Field name           ==>      (Blank = entity list)
Edit business name   ==> NO   (Yes or No)
Edit text            ==> NO   (Yes or No)

Create values        ==> NO   (Yes or No)
IF YES:
  Edit text          ==> NO   (Yes or No)

Help source file name:
==> C:\TMP\APSEXT

```

```

COMMAND ==> _
Specify the items to be included in the extract, then press ENTER

Screen name          ==>          (Blank for entity list)
Edit business name   ==> NO      (Yes or No)
Edit text            ==> NO      (Yes or No)

Local fields         ==> NO      (Yes or No)
IF YES:
  Edit business name ==> NO      (Yes or No)
  Edit text          ==> NO      (Yes or No)

Create values        ==> NO      (Yes or No)
IF YES:
  Edit text          ==> NO      (Yes or No)

Help source file name:
==>> C:\TMP\APSEXT

```

USERNAME

Category: S-COBOL structure (see *S-COBOL Structures*)

Description: Direct either the current APS paragraph name or one you designate to name a procedure generated by the APS Precompiler, so that your program follows the conventions, requirements or preferences of a given application or installation.

Syntax: USERNAME *paragraphname*

- Comments:**
- To activate any USERNAME statement, select the USERNAME option at compile time from the APS Precompiler Options screen; otherwise, the paragraph name for a generated procedure is G--nnn, where G means generated and nnn begins with 001 and increases by one for each generated paragraph name in top-down order.
 - When USERNAME is selected, APS creates paragraph names as follows.
 - APS generates single dashes instead of double dashes.

- APS uses the current paragraph name, unless USERNAME is coded in a paragraph. Then it uses that paragraph name until the end of the paragraph.
- USERNAME paragraph applies only to names assigned for procedures generated as the direct result of code within the paragraph where USERNAME appears. When more than one USERNAME is coded in a paragraph, APS assigns the first user-designated paragraph name until it is overridden by a second name in the APS processing sequence.
- USERNAME does not appear in the generated code.
- Be careful to avoid any naming conflicts that can arise with USERNAME.

VALUE (Data Structure)

Category: Data Structure Painter construct (see *Data Structures*)

Description: Code VALUE clauses in your data structures.

Syntax: *dataname PICformat*
 [...] [VALUE|V] 'valueclause'

- Comments:**
- Valueclause can be as long as you need, continuing on subsequent lines with the continuation symbol.
 - You can code valueclause on the same line as dataname, if it fits entirely on that line.
 - Do not enclose numeric values in single quotation marks.

Examples: Data Structure Painter format:

```
-LINE- ----- Data Structure Painter -----
000001 WRK1-FIELD-7 X(120)
000002 ... VALUE 'A LONG LITERAL MAY BE
000003 ... CONTINUED ON ONE OR
000004 ... MORE LINES'
000005 WRK1-FIELD-8 X(80)
000006 ... V 'A VALUE CLAUSE THAT SPANS
```

```
000007 ... TWO OR MORE LINES MUST BEGIN
000008 ... ON ITS OWN LINE'
```

Generated COBOL code:

```
01 WRK1-FIELD-7 VALUE 'A LONG LITERAL MAY BE CONT
    'INUED ON ONE OR MORE LINES'
    PIC X(120).
01 WRK1-FIELD-8 VALUE 'A VALUE CLAUSE THAT SPANS
    'TWO OR MORE LINES MUST BEGIN ON ITS OWN LINE'
    PIC X(80).
```

Data Structure Painter format:

```
-LINE- ----- Data Structure Painter -----
000001 WRK1-FIELD-5 x(18) V'18 CHARACTERS LONG'
000002 WRK1-FIELD-6 x(13)
000003 ... V'13 CHARS LONG'
```

Generated COBOL code:

```
01 WRK1-FIELD-5 PIC X(18)
    VALUE '18 CHARACTERS LONG'.
01 WRK1-FIELD-6 VALUE '13 CHARS LONG'
    PIC X(13).
```

VALUE (Report Writer)

Category: Report Writer clause (see *Report Writer Structures* and the *APS User's Guide* chapter *Create Reports with Report Writer*)

Compatibility: Batch environments

Description: Designate, as literals, any values that can be interpreted as PIC characters, such as embedded Xs and 9s to a mock-up text field.

APS considers two or more consecutive COBOL PIC characters a COBOL PIC clause and all other strings as literals, with the following exceptions.

- The letter A is a literal character
- A single hyphen is a COBOL picture character; a string of hyphens is a literal.

- A 9 or an X bounded by spaces is a COBOL PIC; any other character surrounded by spaces is a literal.
- Lower case letters are literals, not I/O fields.

To distinguish literals from PIC strings, such as the literal EXXON, which would be interpreted as the literal E, followed by the PIC string XX, and the literal ON, paint the word as a data field in the report mock-up and use the VALUE clause when coding the detail line data item description in your program.

Syntax: VALUE "*characterstring*" [PIC *picclause*]
[DATA-NAME [IS] *fieldname*]

**Keywords/
Parameters:**

"*character
string*"

Designate a literal as follows.

- Enter the text field on the mock-up as a data field (that is, a series of Xs).
- Code VALUE "*characterstring*", where *characterstring* is the literal value of the text field.

DATA-NAME fieldname Name a sum accumulator established by a SUM or REFERENCE clause. Do not define *fieldname* in Working-Storage. At generation, APS inserts *fieldname* after the level number in the generated report group. DATA-NAME moves the value of the internal SUM accumulator to *fieldname*.

Code DATA-NAME when a SUM UPON clause references DETAIL report group, when the program references a sum accumulator, or when a sum accumulator requires a data name for qualification.

PIC clause

Specify the format of *characterstring*.

Example: Create the report heading QUARTERLY REPORT FOR EXXPRT.

```
====
QUARTERLY REPORT FOR XXXXXXXX
====
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
```

```

01    TYPE IS PAGE HEADING.
      MOCKUP LINE 1
      VALUE 'EXXPERT' PIC X(7)

```

Values, Conversion Values, and Value Ranges

Category: Screen Painter feature (see *Field Edits*)

Description: Ensure that the end user enters only certain values in a screen field by assigning values and value ranges or conversion values to that field.

Note: You can assign values or conversions, but not both.

Procedure: To assign values to a field, follow these steps.

- 1 From the Screen Painter, access the Field Edit Facility.
- 2 Access the Values or Conversions screen by selecting the Values or Conversions prompt on any Field Edit screen.
- 3 Assign a specific value or a range of values using the following syntax formats.

```

value
lowvalue TO|THRU highvalue
lowvalue UP
highvalue DOWN

```

- 4 Assign conversion values using the following syntax format.

```

(input1, input2, ..., inputN, I=internalvalue,
O=outputvalue)

```

- 5 To specify that the listed conversions are the only valid input values, select the Verify Conversion Values option.

Comments: Follow these rules when you specify values.

- Separate each entry with a comma.
- Do not code commas in an entry.

- Code only numeric values or ranges for numeric fields.
- Ensure that the value length meets the restrictions of the Internal Picture format.
- Use negative values only if the internal data representation is signed.
- Use decimals only if the internal data representation includes a decimal.

For example:

```
1000 to 30000, 50 down
NORTH, SOUTH, EAST, WEST
25.5 to 35.5, 45.5 to 55.5, 100
-10 to 25, 50.5 up
```

Follow these rules when you specify conversions.

- Separate each entry with a comma.
- Code the internal storage conversion format in internalvalue.
- Code the output display in outputvalue.
- Ensure that the internalvalue and outputvalue are valid input data.

For example, the entry (ja,jan,i=1,o=January) means that the end user can enter ja or jan; APS converts and stores the ja or jan to 1; and APS displays the output as January.

Variable Length File Support

Compatibility: VSAM batch target

Description APS locates a variable length record description directly under the associated FD and places the fixed length record descriptions in Working-Storage.

WRITE ROUTINE

Category: Report Writer clause (see *Report Writer Structures* and the *APS User's Guide* chapter *Create Reports with Report Writer*)

Compatibility: Batch environments

Description: Override the COBOL WRITE statement and execute your own routine.

Syntax: WRITE ROUTINE [IS] *paragraphname*

- Comments:**
- If you code WRITE ROUTINE, you can define your report record as a group-level record in the File Section and reference it in your WRITE ROUTINE paragraph.
 - APS generates a single-field 01-level report record when you code REPORT IS in the FD statement. You can replace REPORT IS with a definition of a group-level record--that is, an 01-level record with elementary data items. APS generates a 248-byte 01-level report record in Working-Storage, for the largest possible report; it generates a 250-byte record if the RED keyword CODE clause is coded.
 - If FDs are not included in your program, as when accessing GSAM files, you can still create a WRITE routine by defining an 01-level user file record. Report Writer uses this record to generate an 01-level report file record in Working-Storage that stores the report records.

Example:

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
FD      INPUT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 0 RECORDS.
01      PART-STOCK-REC                                PIC X(80).
FD      REPORT-OUTPUT-FILE
        LABEL RECORDS ARE STANDARD
01      USER-REPORT-RECORD.
        03  USER-REPORT-APS-PART                    PIC X(248).
        03  USER-REPORT-USER-PART                   PIC X(7).
        .
RED     STOCK-REPORT
        .
        WRITE ROUTINE IS USER-DEFINED-PARA
```

```

      .
      NTRY
      .
      PARA  USER-DEFINED-PARA
      .
      MOVE STOCK REPORT RECORD TO
      ...USER-REPORT-APS-PART

```

WS

Category: Program Painter and Specification Editor keyword (see *Keywords*)

Description: Define or include data structures in the Working-Storage Section.

Syntax: -KYWD- 12-*----20---*----30---*----40---*----50---*----60
 WS
 kywd *associated data structure*

Comment: Associated data structure keywords are *01*, *DS*, *REC*, or *++*.

Example: Use Section keywords to code Working-Storage data structures.

```

-KYWD- 12-*----20---*----30---*----40---*----50---*----60
IO      INPUT-FILE ASSIGN TO UT-S-INPUT
IO      OUTPUT-FILE ASSIGN TO UT-S-OUTPUT
SPNM    C01 IS TOP-OF-PAGE
FD      INPUT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 0 RECORDS
01      INPUT-REC          PIC X(80).
DS01    INPUT-REC
FD      OUTPUT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 0 RECORDS
REC     OUTPUT-REC        X80
01      OUTPUT-REC-R REDEFINES OUTPUT-REC.
        ... COPY OUTREC.

WS
REC     WS-INPUT-REC
        WS-IN-PART-NO      N8
        WS-IN-DESC        X50
        WS-IN-BASE-PRICE  N6V2

```

01 WS-OUT-REC
 DS05 WSOUTREC

XCTL

Category: Data communication call (see *Data Communication Calls*)

Compatibility: CICS, ISPF Dialog, and ISPF Prototyping, targets

Description: Transfer control from a program at one logical level to an APS or non-APS application program at the same level, and pass the Commarea.

Syntax:

CICS Transferring to a APS program:

```
[TP-]XCTL programname [errorpara]
... [LENGTH(value)]
... [DLIUIB pcbname [pcbname ...]]
... [userparm [userparm] ...]
```

Transferring to a non-APS program:

```
[TP-]XCTL programname(NONAPS) [errorpara]
... [LENGTH(value)]
```

ISPF Prototyping

```
[TP-]XCTL programname[ (NONAPS) ]
... [LENGTH(value)]
```

Parameters:	DLIUIB <i>pcbname</i>	DLI interface block and the Program Control Block required by the next program.
	<i>errorpara</i>	User-defined error routine to perform when an abnormal condition occurs. <i>Errorpara</i> is positional; if omitted, code an asterisk (*) in its place.
	LENGTH(<i>value</i>)	Maximum length of data; can be a literal or COBOL data name defined as S9(04)COMP. Can also be a partial length.

(NONAPS)	The program is not an APS program.
NOTERM	Do not terminate the screen display for the calling program, that is, display screens for both the called and the calling program.
<i>programname</i>	Program name; can be a literal, variable, or combination. If you precede a variable name with a slash (/), APS moveS the literal to it.

- Comments:**
- When transferring control to an APS program, XCTL transfers control to the program specified in the call and passes the TP-COMMAREA. Ensure that the Commarea in the transferred-to program is the same length as the TP-COMMAREA you pass.
 - When transferring control to a non-APS program, XCTL transfers control to the specified program and passes the TP-USERAREA. Ensure that the Commarea in the transferred-to program is the same length as the TP-USERAREA you pass. (TP-USERAREA gets its value from &TP-USER-LEN.) Additionally, code the following before your NTRY statement.

```
% &TP-PROGRAM-INVOCATION = "NONAPS"
```

ISPF Dialog

XCTL, which calls a subprogram at the next lower level, operates identically to LINK. XCTL invokes LINK, providing no additional functionality and does not pass the COMMAREA; XCTL is provided for upward compatibility.

- Examples:** Transfer control to PGM001. There is no active PSB and no passing of arguments.

```
XCTL PGM001
```

Transfer control to the program whose name is stored in WS-PROGNAME. There is no active PSB and no passing of arguments.

```
XCTL /WS-PROGNAME
```

Move the value PGM003 to WS-PROGNAME and transfer control to that program. There is no active PSB and no passing of arguments.

```
XCTL PGM003/WS-PROGNAME
```

Transfer control to PGM004; execute an error routine for an abnormal condition. There is no active PSB and no passing of arguments.

```
XCTL PGM004 ERR-PARA
```

Transfer control to PGM005. A PSB is scheduled and the transferred-to program uses PCB ABC-PCB.

```
XCTL PGM005 * DLIUIB ABC-PCB
```

Transfer control to a non-APS program, PROG001.

```
XCTL PROG001(NONAPS)
```


Index

Symbols

++ keyword 11
/* keyword 75

Numerics

01 keyword 11
 use in Report Writer 11, 405
66-level data item 111
66-level number 13
77-level data item 111
77-level number 264
88-level data item 111
88-level number 14
88-level status flags 225

A

abbreviated syntax for S-COBOL 417
abnormal condition processing
 IMS DB 227
 IMS DC 231
 SQL 233
 VSAM batch 235
 VSAM online 237
AB-ON-DC-CALL flag
 IMS DC 230, 231
AB-ON-REC flag
 IDMS DB 225
 IMS DB 227

 SQL 232, 233
 VSAM batch 234
 VSAM online 236, 237
ABORT keyword
 TP-BACKOUT 462
ABRT checkpoint 288
accumulators, Report Writer
 initializing 305
 page 305
 sum 305, 380, 438, 447
addressability, CICS 67
After DB Access control point 91
After Loop control point 91
After-Enter-Check control point 89
After-Receive-Para control point 89
ALARM keyword
 DLG-SETMSG 202
ALL keyword
 DB-CLOSE 121
 DB-ERASE 131
 DB-FREE 136
 DB-OPEN 158
 IDM-COMMIT 282
alternate values
 DB-DECLARE 124
 DB-ERASE 130
 DB-MODIFY 139
 DB-OBTAIN 148
 DB-PROCESS 164
 DB-STORE 176
ALTRESP keyword
 MSG-SW 338
 SEND 432
ALWAYS flag 475
AND keyword
 DB-DECLARE 125

- DB-ERASE 131
- DB-MODIFY 139
- DB-OBTAIN 149
- DB-PROCESS 164
- DB-STORE 177
- APCICSIN control file 80
- APDB2IN control file 85
- APDLGIN control file 85
- APFEIN control file 84
- APHLPIN control file 88
- APIMSIN control file 85
- Application Definition Report (AP01) 16, 407
- application field edit routines 18, 251
- Application Painter
 - accessing 24
 - generating reports 32, 407
 - user processing exits 27
- Application Selection screen, user-defined
 - field edits 19
- applications
 - application definition, copying 27
 - application definition, creating 24
 - components of, copying 27
 - components of, deleting 27
- APS structures
 - data communication calls 101
 - database calls 97
 - Report Writer 404
- APSDBDC file 81
- APS-EDITS-PASSED flag 344, 347
- APS-END-PROCESS flag 169
 - VSAM batch 258
 - VSAM online 259
- APSMACS library 26
- APS-MSG-EDIT-ERROR flag 344, 346
- APS-MSG-IO-ERROR flag 344, 347
- APS-PROCESS-CTR flag 169
- APS-ROW-SUB data field 21
- APS-VSAM-NUMREC field 259
- APVSAMIN control file 87
- AREA keyword
 - DB-OBTAIN 149
 - DB-OPEN 158
 - DB-PROCESS 164
- arguments
 - passing 359, 472
 - performing a paragraph with 359
 - positional 106
- AS parameter
 - DLG-VDEFINE 204
- assembler macros, screen generation parameter 273
- assign
 - see CIC-ASSIGN call
- associated program, ISPF prototyping generation option 276
- AT END condition 39, 296
- ATTR call 40
- attributes, field
 - assigning to a specific field 42
 - blinking 275
 - color 44, 275
 - cursor position, initializing 44
 - intensity 43, 275
 - light pen detection 44
 - list of 42
 - modified data tag 43
 - modified data tags, CICS 336
 - modifying at run time 40, 44
 - numeric keyboard locking 44, 45
 - overriding 40
 - protected 274
 - protected, ISPF prototype 45
 - resetting 71, 471
 - reverse video 275
 - underlining 275
 - unprotected 274
- AVG function 246
 - SQL 265

B

- BACKOUT
 - see TP-BACKOUT call
- Before DB Access control point 90
- Before Loop control point 90

Before-Send-Para control point 89

BIND

see DB-BIND call

bind options, SQL 46

blank lines

suppress in output 81

BLANK WHEN ZERO Report Writer clause 438

BLL cells 469

BMS mapsets

first line of, setting 276

generated name 345

multiple-map 426

names, overriding 275

BROWSE keyword

DB-PROCESS 164

bypass field edits 84, 240

bypassing field edits 84

C

CA keyword 49, 464, 465, 466

calling subroutines/subprograms 321

CANCEL keyword

NTRY 342

canceling

see CIC-CANCEL call

canceling processing 462

CCODE parameter

DB-MODIFY 50

DB-OBTAIN 50

DB-PROCESS 50

DB-STORE 50

CHANGE INDICATE Report Writer clause 438

changing 138

see modify

CHAR function 246

CHAR function, SQL 265

Checkin screen, ENDEVOR Interface 51

checking in files

to ENDEVOR 51

checking out files

from ENDEVOR 53

Checkout screen, ENDEVOR Interface 53

checkpoint/restart 454

CIC-ADDRESS call 55

CIC-ASSIGN call 55

CIC-CANCEL call 56

CIC-DELAY call 57

CIC-DELETEQ-TD call 58

CIC-DELETEQ-TS call 59

CIC-FREEMAIN call 59

CIC-GETMAIN call 60

CIC-LOAD call 61

CIC-READQ-TD call 62

CIC-READQ-TS call 63

CIC-RELEASE call 64

CICS

addressability 67

addressability of Linkage Section records
468

APCICSIN control file 80

assigning LOW-VALUES to fields 471

BMS mapsets, for a program 426

BMS mapsets, names, overriding 275

calls, list of 102, 106

clearing screen fields 71

Commarea, passing 323, 463

conversational programming 463

data transmission 336

EIBRCODE flags 222, 236

error handling 222, 236

Exceptional Conditions 222, 236

generating program template 340

invocation modes 256

ISI-ERRORS 236

main storage area 55, 59, 60, 64

modified data tag, setting 43

NTRY customization exits 345

overriding attributes 40

passing the Commarea 497

PCB use with 375

performing paragraphs 460, 472

PF keys 411

prototyping under ISPF 27

PSB use with 375

receiving multiple screens 426

- resetting attributes 71
- sending screen data 431
- specifying as target 24, 118
- status flags 236
- syntax parameter 83
- task processing starting 67
- task processing, stopping 57
- temporary storage area 59, 63, 70
- terminating processing 462
- transaction ID, specifying 275
- transferring program control 321, 497
- transient data 58, 62, 69
- CICS options, use with
 - CIC-ADDRESS 56
 - CIC-SEND-TEXT 66
 - CIC-START 68
 - CIC-WRITEQ-TS 70
- CIC-SCHEDULE-PSB call 65
- CIC-SEND-TEXT call 66
- CIC-SERVICE-RELOAD call 67
- CIC-START call 67
- CIC-TERM-PSB call 68
- CIC-WRITEQ-TD call 69
- CIC-WRITEQ-TS call 70
- CKEYED keyword
 - DB-OBTAIN 149
 - DB-PROCESS 165
 - DB-STORE 177
- CLEAR call 71
- CLEAR-ATTRS call 71
- clearing screen fields 71
- CLOSE
 - see DB-CLOSE
- COBOL
 - coding in Program Editor 356
 - coding in program or stub 264
 - coding in Specification Editor 88
 - differences from S-COBOL 413
 - edit masks in data structures 109
- COBOL/
 - EVALUATE 243
 - limits 319
- COBOL/2 72
- CODE Report Writer clause 74, 404
- coding rules
 - data communication calls 106
 - database calls 99
 - Report Writer 405
 - S-COBOL programs 415
- color, screen fields 44, 275
- column functions
 - see functions, SQL
- command codes
 - see CCODE keyword
- COMMAREA keyword
 - LINK 322
- comments
 - in data structures 75
 - in generated source code 81
 - in macros 75
 - in program code 75
- COMMIT
 - see DB-COMMIT call and IDM-COMMIT call
- COMMIT generation
 - ISPF 85, 122
- compiling
 - COBOL II compiler, specifying 272
- Component List Report (MS01) 76, 407
- computational formats in data structures 110
- COMT checkpoint 282
- conditional processing with S-COBOL 415
 - AT END 39, 296
 - EVALUATE statement 242
 - FALSE statement 473, 475
 - IF structure 294
 - INVALID KEY 39, 296
 - NEXT SENTENCE 243, 296, 298, 431
 - ON SIZE ERROR 296, 297
 - REPEAT statement 383
 - TRUE statement 473
 - TRUE/FASE/ALWAYS/NEVER flags 475
 - UNTIL statement 383, 483
 - WHILE statement 383, 483
- CONNECT 283
 - see IDM-CONNECT call
- CONT keyword
 - MSG-SW 338

- SEND 432
- CONTCOND keyword
 - MSG-SW 338
 - SEND 432
- continuation
 - data structures 111
 - DB calls 99, 405
 - DC calls 106
 - IDMS DB pass-through support 289
 - S-COBOL structures 416
 - SQL pass-through support 442
- CONTINUE keyword
 - IDM-ROLLBACK 288
 - SEND 432
- control breaks for reports 78, 269, 379, 475
- control files 80
 - CICS 80
 - DB/DC macros 81
 - DB/DC rules 81
 - field edits 84
 - IMS DB and DC 85
 - ISPF Dialog 85
 - SQL 85
 - User Help 88
 - VSAM 87
- CONTROL FOOTING (CF) keyword 477
- CONTROL HEADING (CH) keyword 477
- control points
 - Online Express 88
 - standard, locations in programs, illustration 91
- CONTROL Report Writer clause 78, 379, 404
- conversational programming
 - CICS 463
 - IMS DC 465
 - ISPF Dialog 466
- conversion values, field edits 493
- converting
 - see data conversion values
- COPY
 - see DLG-VCOPY call
- COPY statement 12, 378
- copying
 - application definitions 27
 - with COBOL COPY statement 12, 378
- copylibs/copybooks
 - customize, SQL 194
 - including in programs 304, 452
 - placement 83, 452
- correlation name
 - DB-DECLARE 125
 - DB-OBTAIN 149
 - DB-PROCESS 165
- COUNT built-in function, SQL 279
- COUNT function 246
 - SQL 265
- counters, Report Writer 305
- Create Like function 27
- CURRENCY keyword 287
- currency, establish 143
- currency, establishing 143
- CURRENT keyword
 - DB-OBTAIN 149, 154
 - DB-PROCESS 165
- CURRENT special registers 440
- cursor feedback, IMS DC generation option 276
- CURSOR keyword
 - DB-CLOSE 121
 - DB-FETCH 135
 - DB-OPEN 158
- cursor naming 125, 165
- cursor processing
 - adding rows 175
 - closing 120, 160
 - declaring 122, 160
 - deleting rows 128
 - looping 160
 - modifying rows 138
 - opening 158, 160
 - processing rows 160
 - reading rows 134
 - retrieving rows 134
 - selecting rows 143, 160
 - storing rows 175
 - writing rows 175
- cursor, positioning on screen field 44

- Customization Facility
 - limits 319
- Customization Facility macros
 - comments 75
- Customizer rules
 - including rule libraries 304, 452
 - program locations, specifying for source 452
 - SY* keywords 452
- customizing
 - Online Express programs, database call error processing 88
 - Online Express programs, predefined functions 88

D

- data communication calls 101
 - CICS 102
 - coding rules 106
 - continuation 106
 - control files 81
 - IMS DC 103
 - IMS Fast Path 299
 - ISPF Dialog 104
 - targets supported 101
- Data Element Facility
 - specifying for Project and Group 377
- Data Structure Definition Report (DS01) 107, 407
- Data Structure Painter 109
- data structures
 - 66-level 13
 - 77-level 111
 - 88-level 14
 - application limit 320
 - COBOL edit masks 109
 - COBOL Syntax
 - 11
 - comments 75
 - computational formats 110
 - continuation 111
 - creating in program 377
 - edit masks 109
 - indentation 112
 - INDEXED BY clause 349
 - level numbers 112
 - Linkage Section 326
 - naming conventions 25
 - OCCURS clause 349
 - overview 109
 - PICTURE clauses, COBOL 112
 - picture formats 110, 112
 - REDEFINES clause 379
 - renames clause 13
 - specifying in application definition 25
 - specifying to program 209
 - USAGE clause 110
 - VALUE clause 14, 490
 - Working-Storage 496
- data transmission
 - modified data tags, CICS 336
- database calls 97
 - coding rules 99
 - continuation 99
 - generation of, suppressing 450
 - IDMS 97
 - IMS 98
 - IMS Fast Path 301, 303
 - IMS GSAM 280
 - SQL 98
 - targets supported 97
 - VSAM online 99
- database calls, Online Express
 - control files 81
 - customizing 88
 - error handling 88
- database functions, Online Express
 - customizing 88
- DATA-NAME Report Writer clause 380, 438, 447, 491
- DATE
 - function, SQL 246, 265
 - special register, SQL 440
- date field edits
 - input 114

- internal picture 115
 - output 114
- DB target
 - specifying 24, 118
- DB/DC macros
 - control files 81
- DB2
 - specifying as target 118
- DB2-DEADLOCK flag
 - SQL 232
- DB-BIND call 119, 286
- DB-CLOSE call 120
- DB-COMMIT call 121
- DB-DECLARE call 122
 - DB2 unions 480
 - expressions 246, 265
 - functions 246, 265
 - GROUP BY clause 278
 - HAVING clause 278
 - joining tables 307
 - special registers 440
- DB-ERASE
 - customizing exits 133
- DB-ERASE call 128
 - exit points 244
 - VSAM batch fields 258
 - VSAM online fields 259
- DB-FETCH call 126, 134, 172
- DB-FREE call 135
- DB-GET call 137
- DBKEY keyword
 - DB-OBTAIN 149
- DB-LOOP-MAX keyword 165
- DB-MODIFY call 138
 - exit points 244
 - VSAM online fields 259
- DBNAME keyword
 - DB-BIND 119
- DB-OBTAIN call 143
 - DB2 unions 480
 - exit points 244
 - expressions 246, 265
 - functions 246, 265
 - joining tables 307
 - special registers, SQL 440
 - VSAM batch fields and flags 258
 - VSAM online fields and flags 259
- DB-PROCESS
 - customizing exits 172
- DB-PROCESS call 160
 - DB2 unions 480
 - exit points 244
 - expressions 246, 265
 - functions 246, 265
 - GROUP BY clause 278
 - HAVING clause 278
 - joining tables 307
 - loop counters 169
 - looping 169
 - special registers, SQL 440
 - VSAM batch fields and flags 258
 - VSAM online fields and flags 259
- DB-PROCESS-ID clause 165, 169, 171, 172
- DB-ROLLBACK call 174
- DB-STORE call 175
 - exit points 244
 - VSAM batch fields 258
 - VSAM online fields 259
- DB-SUBSCHEMA call 180
- DC target
 - specifying 118
- DC target, specifying 24
- DDI statements 182
 - VSAM, for KSDS fixed length files 190
 - VSAM, for KSDS variable length files 191
- DDI, SQL
 - copybooks, customize 194
- DDIFILE Report (DB01) 181, 407
- DDN keyword
 - DB-ERASE 131
 - DB-FREE 136
 - DB-MODIFY 139
 - DB-OBTAIN 149
 - DB-PROCESS 165
 - DB-STORE 177
- DDS
 - performing paragraphs 472
 - receiving multiple screens 426

- debugging programs
 - SCBTRACE option 364
- decision table 242
- DECL keyword 199
- Declarative Section
 - DECL keyword 199
 - DPAR keyword 207
 - USE BEFORE REPORTING 485
- declaratives, Report Writer 485
- DECLARE Customization Facility structure
 - limits 319
- declaring table rows 122
- DEFINE
 - see DLG-VDEFINE call
- delay
 - see CIC-DELAY call
- DELETE
 - see DLG-VDELETE call
- deleteq-TD
 - see CIC-DELETEQ-TD call
- deleting
 - application components 27
 - database and file records 128
 - table rows 128
- DEPENDING ON in data structures 349
- DETAIL (DE) keyword 477
- detail lines, Report Writer 475
- detail reports, Report Writer 269, 475
- device type, IMS DC generation option 276
- DFS0AER
 - enable use of 82
- DIF-DOF name, IMS DC generation option
 - 277
- DISCONNECT 284
- DISTINCT keyword
 - DB-DECLARE 125
 - DB-OBTAIN 149
 - DB-PROCESS 165
- DLG
 - conversational programming 466
 - error handling 434
- DLG-AUTO-VARIABLE-VDELETE 205
- DLG-ISPEXEC call 200
- DLG-ISREDIT call 201
- DLG-SETMSG call 201
- DLG-VCOPY call 203
- DLG-VDEFINE call 204
- DLG-VDELETE call 205
- DLG-VREPLACE call 206
- DLG-VRESET call 207
- DLIUIB keyword
 - LINK 322
 - XCTL 497
- Documentation Facility
 - summary of reports 32, 407
- DPAR keyword 207
- DS keyword 209
- DSCA, IMS DC generation option 278
- DSIDERR error flag/condition 236
- DUPKEY error flag/condition 236
- DUP-ON-REC flag
 - IDMS DB 225
 - IMS DB 227
 - SQL 232
 - VSAM batch 234
 - VSAM online 236
- DUPREC error flag/condition 236

E

- Ed-Error-Pre-Send control point 89
- edit masks in data structures 109
- edits, field
 - APFEIN control file 84
 - bypassing 84, 240
 - control file for 84
 - conversion values for 493
 - date fields, input 114
 - date fields, internal picture 115
 - date fields, output 114
 - error handling 239
 - European format for numeric fields 84
 - input, date fields 114
 - input, time fields 117
 - internal picture, date fields 115
 - internal picture, time fields 116

- invalid data, allowing or disallowing 343
- list of 251
- output, date fields 114
- output, time fields 117
- overview of 251
- performing 340
- time fields, input 117
- time fields, internal picture 116
- time fields, output 117
- user-defined 251
- user-defined, control file switches 84
- user-defined, creating 18
- user-defined, referencing APS-generated field names in 19
- user-defined, selecting from predefined list 19
- value ranges for 493
- edits, fields
 - bypassing 84
 - European format for numeric fields 84
 - user-defined, control file switches 84
- EIBRCODE flags, CICS 222, 236
- ENDBR keyword
 - DB-FREE 136
- ENDCONV keyword
 - MSG-SW 338
 - SEND 433
- ENDEVOR, APS Interface
 - checking in files 51
 - checking out files 53
- ENDFILE error flag/condition 236
- END-ON-REC flag
 - IDMS DB 225
 - IMS DB 227
 - SQL 232
 - VSAM batch 234
 - VSAM online 236
- END-PROCESS flag 169
- ENDWHERE keyword
 - DB-MODIFY 139
- Entity Content Report (MS02) 210, 407
- Entity Cross Reference Report (MD01) 212, 407
- Entity Parts List Report (EN01) 214, 407
- Entity Search Utility Report (GS01) 216, 407
- Entity Use Report (EN02) 219, 407
- ENTRY parameter
 - CIC-LOAD 61
- ENTRY statement 221
- ERASE
 - DB-ERASE call 128
- error handling
 - CICS 222, 236
 - field edits 84, 239
 - field edits, bypassing 240
 - IDMS DB 225
 - IMS DB, enable use of DFS0AER 82
 - IMS DC 230
 - IMS DC, enable use of DFS0AER 82
 - ISPF Dialog 342, 343
 - ISPF prototyping 342, 343
 - SAGE-TRACE-FLAG 233
 - SCBTRACE option 364
 - SQL 232
 - VSAM batch 234
 - VSAM online 236
- ERROR parameter
 - CIC-CANCEL 56
 - CIC-DELAY 57
 - CIC-DELETEQ-TD 58
 - CIC-DELETEQ-TS 59
 - CIC-GETMAIN 60
 - CIC-LOAD 61
 - CIC-READQ-TD 62
 - CIC-READQ-TS 63
 - CIC-RELEASE 65
 - CIC-SEND-TEXT 66
 - CIC-START 68
 - CIC-WRITEQ-TD 69
 - CIC-WRITEQ-TS 70
- Error Status control point 90
- ERROR TRACE
 - see SET Customization Facility statements
- errorpara parameter
 - LINK 323
 - MSG-SW 338
 - NTRY 343, 347
 - SEND 432

Error-Send-And-Quit control point 90
 ESCAPE statement 241
 ESDS support
 VSAM batch 259
 VSAM online 261
 European format for numeric fields 84
 EVALUATE statement 242, 321
 COBOL/ 243
 evaluation brackets 75, 81
 Exception Status control point 90
 EXCLUSIVE keyword
 DB-OBTAIN 149
 DB-PROCESS 165
 exit points 244
 EXIT PROGRAM statement 245
 exits, customizing
 DB-ERASE 133
 DB-PROCESS 172
 NTRY 345, 346
 EXPRESS keyword
 MSG-SW 338, 371
 SEND 433
 expressions 246
 SQL 246, 265
 extended attributes 42
 modifying at run time 45
 modifying at run time, ISPF prototyping
 274

F

FALSE flag 475
 FALSE statement 473
 Fast Path
 DC calls 299
 IM-CHKP 454
 IM-CHNG 299
 IM-CMD 299
 IM-DEQ 454
 IM-FLD 301
 IM-FSA 301
 IM-GCMD 299
 IM-GN 299
 IM-GSCD 454
 IM-GU 299
 IM-ISRT 299
 IM-LOG 454
 IM-POS 303
 IM-PURG 299
 IM-ROLB 454
 IM-ROLL 454
 IM-STAT 454
 IM-XRST 454
 system service calls 454
 FD keyword 74, 247
 use in Report Writer 247, 404
 FD parameter
 use in Report Writer 74
 FETCH
 see DB-FETCH call
 FETCH ONLY keyword
 DB-DECLARE 125
 DB-PROCESS 166
 Field Attributes screen 42
 field edits
 control file 84
 Field Edits Report (ED01) 407, 420
 Field/Screen Cross Reference Report (SC02)
 262, 407
 fields
 APS-VSAM-NUMREC 259
 CICS TP-USERAREA 49, 463
 CICS, TP-USER-LEN 49, 463
 DLG, TP-USERAREA 466
 DLG, TP-USER-LEN 466
 field edit, APS-ROW-SUB 21
 field edit, -ATTR 21
 field edit, -EDIT 20, 21
 field edit, -INPT 20
 field edit, -LEN 21
 IMS DB, error 229
 IMS DC, TP-USERAREA 49, 465
 IMS DC, TP-USER-LEN 49, 465
 ISPF Dialog, error 200, 201, 202, 204, 205,
 207
 ISPF Dialog, TP-USERAREA 49, 466

- ISPF Dialog, TP-USER-LEN 49, 466
- Report Writer, internal sum accumulators 380, 438, 447
- Report Writer, PAGE-COUNTER 439
- SQL, null indicator 349
- VSAM batch 258
- VSAM online 259
- FILE keyword
 - DB-CLOSE 121
 - DB-OPEN 158
- File Section
 - file description keyword 247
 - File-Control statement keyword 305
 - Report Writer 247
- FINAL keyword 78, 477
- FIRST keyword
 - DB-OBTAIN 149
 - DB-PROCESS 166
- flags
 - ALWAYS 475
 - APS-END-PROCESS, VSAM batch 258
 - APS-END-PROCESS, VSAM online 259
 - CICS, EIBRCODE 222, 236
 - CICS, error 236
 - CICS, invocation mode 256
 - FALSE 473, 475
 - field edits 19
 - IDMS DB, error 225
 - IMS DB, error 227
 - IMS DC, error 230
 - IMS DC, invocation mode 257
 - ISPF Dialog, invocation mode 258
 - loop counters, DB-PROCESS 169
 - loop counters, LOOP-LIMIT 81, 171
 - looping, DB-PROCESS 169
 - NEVER 475
 - NTRY, error 344, 345, 347
 - NTRY, I/O 344, 345, 347
 - received screen 427
 - RESET-OBTAIN, VSAM batch 258
 - RESET-OBTAIN, VSAM online 259
 - SAGE-TRACE-FLAG 233, 364
 - S-COBOL 415
 - SQL, error 232
 - TP-LINK-INVOKED 258
 - TP-PROGRAM-INVOKED, CICS 256
 - TP-PROGRAM-INVOKED, IMS DC 257
 - TP-PROGRAM-INVOKED, ISPF Dialog 258
 - TP-SCREEN-INVOKED, CICS 256
 - TP-SCREEN-INVOKED, IMS DC 257
 - TP-SCREEN-INVOKED, ISPF Dialog 258
 - TP-TRANSID-INVOKED, CICS 256
 - TP-TRANSID-INVOKED, IMS DC 257
 - TRUE 473, 475
 - VSAM batch 258
 - VSAM batch, error 234
 - VSAM online 259
 - VSAM online, error 236
- FLENGTH parameter
 - CIC-GETMAIN 60
 - CIC-LOAD 61
- footers, Report Writer 475
- format, character 45
- FP-ERR flag
 - IMS DC 230
- FREE 135
 - see DB-FREE call
- FREEMAIN
 - see CIC-FREEMAIN call
- FRFM keyword 264
- FROM keyword
 - DB-ERASE 131, 133
 - DB-MODIFY 140
 - DB-OBTAIN 150
 - DB-STORE 177
 - DLG-VCOPY 203
- FROM parameter
 - CIC-SEND-TEXT 66
 - CIC-WRITEQ-TD 69
 - CIC-WRITEQ-TS 70
 - DB-STORE 179
- functions
 - SQL 246, 265
 - SQL, DB-DECLARE 265
 - SQL, DB-OBTAIN 265
 - SQL, DB-PROCESS 265
- functions, Online Express program
 - database, predefined, customizing 88

- error processing 88
- teleprocessing, predefined, customizing 88

G

- GENERATE Report Writer statement 405
- General-Pre-Send control point 89
- GENERATE Report Writer statement 79, 269
- generating applications
 - job control cards 307
 - options, generator 271
 - options, IDMS 293
 - options, precompiler 364
 - options, resetting to default values 272
 - SQL options 46
 - suppressing database calls 450
- generating reports 32, 407
- generating screens
 - generation parameters 273
 - generation parameters, for all targets 273
 - generation parameters, for ISPF Dialog 355
 - generation parameters, for KANJI format 274
- Generation Options screen 271
- GENONLY keyword
 - DLG-VCOPY 203
- GENONLY parameter
 - DLG-VDEFINE 204
 - DLG-VREPLACE 206
- GET
 - see DB-GET call
- GETMAIN
 - see CIC-GETMAIN call
- global
 - field edit messages 239
 - stubs, creating 444
 - stubs, keyword 444
- GO TO...DEPENDING ON 26, 243

- GROUP BY clause 278
 - DB-DECLARE 278
 - DB-PROCESS 278
- GROUP INDICATE Report Writer clause 438
- grouping data elements, SQL 278
- GSAM
 - IM-CLSE 280
 - IM-GN 280
 - IM-GU 280
 - IM-ISRT 280
 - IM-OPEN 280

H

- HAVING clause 278
 - DB-DECLARE 278
 - DB-PROCESS 278
- headers, Report Writer 475
- HELP keyword
 - DLG-SETMSG 202
- HOLD keyword
 - DB-COMMIT 122
 - DB-OBTAIN 150, 154
 - DB-PROCESS 166
- HOLD parameter
 - CIC-LOAD 61
 - DB-OBTAIN 82

I

- IDCS keyword 282
- IDM-COMMIT call 282
- IDM-CONNECT call 283
- IDM-DISCONNECT call 284
- IDM-GEN-AUTOSTATUS flag 286
- IDM-IF call 285
- IDM-PROTOCOL call 285
- IDM-RETURN call 287
- IDM-ROLLBACK call 288

- IDMS DB
 - ABRT checkpoint 288
 - binding records 119
 - calls, list of 97
 - closing files and subschemas 120
 - COMT checkpoint 282
 - COPY IDMS statement 119
 - deleting records 128
 - error handling 225
 - modifying records 138
 - moving data to Working-Storage 137
 - native (pass-through) support 282, 288
 - opening 158
 - reading records 143, 144
 - retrieving and processing records 160
 - specifying as target 24, 118
 - status flags 225
 - storing records 175
 - subschemas 180
 - topics, list of 100
 - writing records 175
- IDMS Options screen 293
- IDMS statement 288
- IDMSREC keyword
 - DB-OBTAIN 150
- IDSS keyword 282
- IF 285
- IF structure 264, 294
 - nesting 297
- ILLOGIC error flag/condition 236
- IM-CHKP system service call 454
- IM-CHNG data communication call 299
- IM-CLSE database call 280
- IM-CMD data communication call 299
- IM-DB-PCB-KEY-FEED-BACK error field 229
- IM-DB-PCB-KEY-KFBLEN error field 229
- IM-DB-PCB-SEGLEV error field 229
- IM-DB-PCB-SEGNAME error field 229
- IM-DEQ system service call 454
- IM-FLD database call 301
- IM-FSA database call 301
- IM-GCMD data communication call 299
- IM-GN data communication call 299
- IM-GN database call 280
- IM-GSCD system service call 454
- IM-GU data communication call 299
- IM-GU database call 280
- IM-ISRT data communication call 299
- IM-ISRT database call 280
- IM-LOG system service call 454
- IM-OPEN database call 280
- IM-POS database call 303
- IM-PURG data communication call 299
- IM-ROLB system service call 454
- IM-ROLL system service call 454
- IMS
 - specifying as target 27
- IMS DB
 - abnormal condition processing 227
 - APIMSIN control file 85
 - blocking parameter 83
 - calls, list of 98
 - checkpoint/restart 454
 - command codes 50
 - copylib record parameter 82
 - DDI statements 182
 - deleting records 128
 - enable prototype mode 82
 - error handling 227
 - error handling, error fields 229
 - Fast Path 301, 303, 454
 - GSAM calls 280
 - modifying records 138
 - reading records 143, 145
 - retrieving and processing records 160
 - specifying as target 24, 118
 - status flags 227
 - storing records 175
 - subschemas 180
 - SUPPRESS option 450
 - topics, list of 100
 - writing records 175
- IMS DC
 - abnormal condition processing 231
 - addressability of Linkage Section records 468
 - APIMSIN control file 85
 - assigning LOW-VALUES to fields 471

- blocking parameter 83
- calls, list of 103
- checkpoint/restart 454
- clearing screen fields 71
- conversational programming 465
- cursor feedback, specifying 276
- DDI statements 182
- device type, specifying 276
- DIF-DOF name, specifying 277
- DSCA, specifying 278
- enable prototype mode 82
- error handling 230
- Fast Path 299, 454
- generating program template 340
- invocation modes 257
- lines per page, specifying for printing 278
- MFS function keys, assigning 329
- MID MOD, reordering 332
- MID, default values, specifying 277
- MID, name, specifying 277
- MOD, fill character, specifying 278
- MOD, name, specifying 278
- NTRY customization exits 346
- operator logical paging, specifying 277
- overriding attributes 40
- PCBs 371
- performing paragraphs 472
- PF keys 362
- PF keys, ret 411
- program support variables 257
- prototyping under ISPF 27
- PSBs 371
- receiving multiple screens 426
- sending data or messages 337
- sending messages 339, 431, 434
- specifying as target 24, 118
- SQL considerations 371
- status flags 230
- terminating processing 462
- terminating programs 460
- topics, list of 107
- trancodes, creating 330
- transferring program control 321
- IMSREC keyword
 - DB-ERASE 131
 - DB-MODIFY 140
 - DB-OBTAIN 150, 153
 - DB-PROCESS 166
 - DB-STORE 177
- IM-STAT system service call 454
- IM-SUPPRESS-DB-CALL 450
- IM-XRST system service call 454
- INCLUDE Customizer statement 304
- INCLUDE PANVALET member 11
- including in programs
 - copybooks 304
 - copylibs/copybooks 452
 - rule libraries 304, 452
- indentation
 - data structures 112
 - DB calls 99
 - DC calls 106
 - S-COBOL programs 357, 413
- INDEXED BY clause in data structures 349
- initializing reports, Report Writer 305
- INITIATE Report Writer statement 305, 405
- INITIMG parameter
 - CIC-GETMAIN 60
- input field edits
 - date fields 114
 - time fields 117
- Input-Output Section 305
- intensity, screen fields 43, 275
- internal picture
 - date fields 115
 - time fields 116
- INTERVAL parameter
 - CIC-DELAY 57
 - CIC-START 68
- INTO keyword
 - DB2 unions 481
 - DB-FETCH 135
 - DB-OBTAIN 150
 - DB-PROCESS 166
- INTO parameter
 - CIC-READQ-TD 62
 - CIC-READQ-TS 63

- DLG-REPLACE 206
- INVALID KEY condition 39, 296
- invocation modes 351
 - CICS 256
 - IMS DC 257
 - ISPF Dialog 258
- INV-ON-REC flag
 - VSAM batch 234
 - VSAM online 236
- INVREQ error flag/condition 236
- IO keyword 305
 - use in Report Writer 404
- IO PCB
 - see PCB
- IOERR error flag/condition 236
- IRQ-ON-REC flag
 - VSAM online 236
- ISPEXEC
 - see DLG-ISPEXEC call
- ISPF Dialog
 - addressability of Linkage Section records 468
 - APDLGIN control file 85
 - assigning LOW-VALUES to fields 471
 - calls, list of 104
 - clearing screen fields 71
 - COMMIT generation 85, 122
 - conversational programming 466
 - displaying screen data 431
 - error handling 342
 - generating program template 340
 - invocation modes 258
 - invoking programs 323
 - overriding attributes 40
 - passing the Commarea 323, 467, 497
 - PF keys 363, 411
 - receiving multiple screens 426
 - resetting attributes 71
 - screen generation parameters 355
 - specifying as target 24, 118
 - terminating programs 460
 - transferring program control 321, 497
- ISPF Panel Options screen 355

- ISPF prototyping
 - displaying screen data 431
 - error handling 342
 - generating program template 340
 - overriding attributes 40
 - passing the Commarea 497
 - PF keys 411
 - receiving multiple screens 426
 - sending data or messages 337
 - terminating programs 460
 - transferring program control 321, 497
- ISREDIT
 - see DLG-ISREDIT call
- ITEM parameter
 - CIC-READQ-TS 64
 - CIC-WRITEQ-TS 70
- iterative expressions, Report Writer 405, 438, 447
- IVD-ON-REC flag
 - VSAM batch 234

J

- job control cards 307
- joining tables
 - DB-DECLARE 307
 - DB-OBTAIN 307
 - DB-PROCESS 307
- JUSTIFIED RIGHT Report Writer clause 438

K

- KANJI
 - setting DBCS 83
- KANJI format
 - generation parameter 274
 - ruled lines 45
 - specifying for fields 45
- KEY INTO keyword 287

- KEYLENGTH keyword
 - DB-ERASE 131
 - DB-OBTAIN 150
 - DB-PROCESS 166
- keywords, Program Editor 311
- KLEN keyword
 - DB-ERASE 131
 - DB-OBTAIN 150
 - DB-PROCESS 166

- L**
- LAST keyword
 - DB-OBTAIN 150
 - DB-PROCESS 166
- LEN keyword
 - DLG-VCOPY 203
- LEN parameter
 - DLG-REPLACE 206
 - DLG-VDEFINE 205
- LENGERR error flag/condition 236
- length
 - screen fields, changing 43
- LENGTH keyword
 - LINK 323
 - XCTL 497
- LENGTH parameter
 - CIC-GETMAIN 60
 - CIC-LOAD 61
 - CIC-READQ-TD 62
 - CIC-READQ-TS 64
 - CIC-SEND-TEXT 66
 - CIC-WRITEQ-TD 69
 - CIC-WRITEQ-TS 70
- level numbers in data structures 13, 112
- light pen detection 44
- limits in APS 319
- line counter, Report Writer 305
- LINE Report Writer clause 477
- LINE-COUNTER Report Writer field 305
- LINK call 321
 - Commarea, passing 323
 - passing the Commarea 323, 463, 467
 - PSB use with 375
- LINKAGE
 - see TP-LINKAGE call
- LINKAGE keyword
 - SCRNLIST 427
- Linkage Section keyword 326
- linking programs and subprograms 321, 498
- LINK-INVOKED
 - see TP-LINK-INVOKED flag
- literals
 - MFS system 45
 - S-COBOL, concatenating 416
- literals, Report Writer 388, 491
- LK keyword 326
- loading
 - see CIC-LOAD call
- Loc(ation) field, Application Painter 26
- local
 - stubs, keyword 444
- locations, program
 - specifying for Customizer source 452
- logical paging 435
- LONG keyword
 - DLG-SETMSG 202
- looping
 - DB-PROCESS call 160
 - loop counters, DB-PROCESS call 169
 - loop counters, LOOP-LIMIT 81, 171
 - REPEAT statement 383
 - UNTIL statement 383, 483
 - WHILE statement 383, 483
- LOOP-LIMIT flag 81, 171
- LOW-VALUES in screen fields 71, 471

- M**
- Macro/Program Cross Reference Report (MC01) 327, 407
- macros, user-defined
 - including in application definition 26
 - naming conventions 26

- program locations for 26
- main storage area, CICS 55, 59, 60, 64
- masks in data structures 109
- MAX function
 - SQL 265
- MDT (modified data tags)
 - CICS 336
- message switching 337
- messages, sending in IMS DC 339, 434
- MFS Function Keys screen 329
- MFS mapsets
 - assigning to PF keys 329
 - system literals 45
 - trancode literal values, specifying 278
 - trancodes, creating 330
- MFS Trancode Construction screen 330
- MID
 - default values, specifying 277
 - name, specifying 277
- MID MOD Reorder screen 332
- MIN function
 - SQL 265
- Misc-User-Paragraphs control point 90
- MOCK keyword 333, 405
- Mock-Up Report (RP01) 335, 407
- MOCKUP Report Writer statement 334, 405
- MOD
 - fill character, specifying 278
- MODE keyword
 - DB-OPEN 159
- modifiable extended attributes
 - prototyping under ISPF 274
- modified data tags
 - CICS 336
 - setting 43
- MODIFY 138
 - see DB-MODIFY call
- MODIFY keyword
 - MSG-SW 371
- modifying
 - database and file records 138
 - table rows 138
- modifying rows and records 138

- moving database records to Working-Storage 143
- MSG-SW call 337
 - IMS PCBs 371

N

- nested IF 297
- NEVER flag 475
- NEXT GROUP Report Writer clause 478
- NEXT keyword
 - DB-OBTAIN 151
 - DB-PROCESS 167
- NEXT parameter
 - CIC-READQ-TS 64
- NEXT SENTENCE 243, 296, 298, 416, 431
- NEXTREC keyword
 - DB-OBTAIN 153
- NOABORT keyword
 - TP-BACKOUT 462
- NOALTRESP keyword
 - MSG-SW 338
 - SEND 432
- NOCA keyword
 - LINK 323
- NOCONT keyword
 - MSG-SW 338
 - SEND 432
- NOCONTINUE keyword
 - SEND 433
- NODENAME keyword
 - DB-BIND 119
- NOENDCONV keyword
 - MSG-SW 338
 - SEND 433
- NOERASE keyword
 - SEND 433
- NOEXPRESS keyword
 - MSG-SW 338
 - SEND 433
- NO-MORE-MSGS flag
 - IMS DC 230

NO-MORE-SEGS flag
IMS DC 230

NONAPS keyword
LINK 323
XCTL 498

NONE keyword
DB-DECLARE 309
DB-OBTAIN 309
DB-PROCESS 309

NOPURG keyword
MSG-SW 339
SEND 433, 434

NOREDEF keyword
SCRNLIST 427

NORETRY keyword
NTRY 339, 343

NORETURN keyword
SEND 433

Normal Status control point 90

NOSPACE error flag/condition 236

NOSUSPEND parameter
CIC-WRITEQ-TS 70

NOTERM keyword
XCTL 498

NOTFND error flag/condition 236

NOTOPEN error flag/condition 236

NTF-ON-REC flag
IDMS DB 225
IMS DB 227
SQL 232
VSAM batch 234
VSAM online 236

NTRY keyword 340, 351, 427
customization exits 345, 346

NULL
see TP-NULL call

null indicators
SQL 349

null values in screen fields 71, 471

numeric keyboard locking 44, 45

NUMITEMS parameter
CIC-READQ-TS 64

O

OBTAIN 143
see DB-OBTAIN call

OBTAIN DB-OBTAIN call see 143

OCCURS in data structures 349

OF keyword
DB-ERASE 131
DB-OBTAIN 151
DB-PROCESS 167
DB-STORE 177

OK-ON-DC-CALL flag
IMS DC 230

OK-ON-REC flag
IDMS DB 225
IMS DB 227
SQL 232
VSAM batch 234
VSAM online 236

ON SIZE ERROR clause 296, 297

Online Express
control points 88
customizing programs, database call error processing 88
customizing programs, predefined functions 88
database calls, customizing 88
limits 319

Online Express paragraphs 356

opening
cursor sets 158
records 158

operator logical paging, specifying 277

operators, relational in S-COBOL 417

OPT keyword 344, 351, 370

OPTIMIZE keyword
DB-DECLARE 125
DB-PROCESS 167

OR keyword
DB-DECLARE 125
DB-ERASE 131
DB-MODIFY 140
DB-OBTAIN 151

- DB-PROCESS 167
- DB-STORE 177
- ORACLE
 - exit points 244
- ORDER BY keyword
 - DB-DECLARE 278
 - DB-PROCESS 278
- ORDER keyword
 - DB2 unions 482
 - DB-DECLARE 126, 310
 - DB-PROCESS 167, 310
- OS4
 - storing records 175
- output field edits
 - date fields 114
 - time fields 117
- OVERPRINT Report Writer clause 352, 405
- OWNER keyword
 - DB-OBTAIN 151

P

- page counter, Report Writer 305
- PAGE FOOTING (PF) keyword 478
- PAGE HEADING (PH) keyword 478
- PAGE LIMIT Report Writer clause 353, 405
- PAGE-COUNTER Report Writer field 439
- PAGE-LIMIT Report Writer option 78
- Painter Menu
 - generating reports 32, 407
- PANVALET keyword 11
- PARA keyword 356, 413
- paragraph names
 - COBOL/ 73
- paragraphs
 - arguments 359, 472
 - in Online Express 356
 - performing 359, 472
 - S-COBOL names 415
- Parm screen, field edits 240
- passing the Commarea
 - CICS 323, 463
 - DDS 323
 - ISPF Dialog 323, 467
- pass-through support
 - IDMS DB 282, 288
 - SQL 441
- PCB keyword
 - DB-ERASE 131, 132
 - DB-MODIFY 140
 - DB-OBTAIN 151, 152, 153
 - DB-PROCESS 167, 168
 - DB-STORE 177
- PCB parameter
 - DB-STORE 179
- PCBs
 - CICS 375
 - IMS DC 371
- PERFORM call 472
- PERFORM statement 358
- performing paragraphs with arguments 359, 472
- PERMANENT keyword
 - DB-ERASE 132
- PF keys
 - assigning MFS functions to 329
 - defined in CICS 361
 - defined in IMS DC 362
 - defined in ISPF Dialog 363
 - RESET call 411
- PIC clause
 - in data structures 110, 112
 - Report Writer 380, 438, 447, 491
- PIC keyword
 - DLG-VCOPY 203
- PIC parameter
 - DLG-REPLACE 206
 - DLG-VDEFINE 205
- PLUS keyword 478
- positional arguments 106
- POS-ON-REC flag
 - IDMS DB 225
 - IMS DB 227
- Post-RB1-Row-To-Recontrol point 90
- Post-Rec-To-RB1-Row control point 90
- Post-Rec-To-Screen control point 89

- Post-Screen-Read control point 89
- Post-Screen-To-Rec control point 89
- Pre-Branch control point 89
- Precompiler Options screen 364
- precompiler, APS
 - options for 364
- predicates, native SQL 126, 132, 141, 152, 168, 178
- Pre-Function-Test control point 89
- Pre-RB1-Row-To-Rec control point 90
- Pre-Rec-To-RB1-Row control point 90
- Pre-Rec-To-Screen control point 89
- Pre-Screen-To-Rec control point 89
- Pre-Term control point 89
- Preventing 84
- PREVIOUS keyword
 - DB-OBTAIN 151
 - DB-PROCESS 167
- printing, suppressing in Report Writer 450
- PROC keyword 351, 370
- Procedure Division
 - NTRY keyword 340
 - PROC keyword 370
 - PROCEDURE DIVISION USING 342, 370, 469
 - Report Writer 269, 305, 461, 485
- PROCESS 160
 - see DB-PROCESS call
- PROCESS-CTR flag 169
- processing
 - database and file records 160
 - table rows 160
- processing rows and records 160
- PROG statement for OPT keyword 352
- program
 - control return to calling program 460
 - control, terminating processing 462
- Program DB/DC Report (PG02) 372, 407
- Program Definition Report (PG01) 374, 407
- Program Editor keywords 311
- program locations for Customizer source 452
- Program Painter
 - DB calls 97
 - DC calls 101
 - Report Writer structures 404
- PROGRAM parameter
 - CIC-RELEASE 65
- program parameter
 - CIC-LOAD 62
- PROGRAM-INVOKED flag
 - see TP-PROGRAM-INVOKED flag
- Program-Invoked-Para control point 89
- programs, batch
 - generation option, Report Writer 364
 - naming conventions 25
 - specifying as target 24
 - specifying in application definition 25
- programs, online
 - naming conventions 25
 - specifying in application definition 25
- Project Group Environment screen 376
- protected fields 274
 - ISPF prototype screens 45
- PROTOCOL 285
- prototype
 - enable IMS prototype mode 82
 - enable VSAM prototype mode 82
 - generation of, suppressing DB calls 450
- prototyping under ISPF
 - associated programs, specifying 276
 - CICS or IMS DC applications 27
 - field names 43
 - modifiable extended attributes 274
 - specifying as target 24, 118
 - suppressing database calls 450
- PSBs
 - naming conventions 26
 - reporting on 372, 407
 - specifying in application definition 26
 - use with CICS 65, 68, 375
 - use with IMS DC 371
- punctuation in S-COBOL programs 416
- PURG keyword
 - MSG-SW 339
 - SEND 433, 434

Q

QUEUE parameter

- CIC-DELETEQ-TD 58
- CIC-DELETEQ-TS 59
- CIC-READQ-TD 63
- CIC-READQ-TS 64
- CIC-WRITEQ-TD 69
- CIC-WRITEQ-TS 70

R

READQ-TD

- see CIC-READQ-TD call

READQ-TS

- see CIC-READQ-TS call

REC keyword 377

- DB-BIND 119
- DB-DECLARE 125
- DB-FREE 136
- DB-GET 137
- DB-MODIFY 140
- DB-OBTAIN 151, 153
- DB-PROCESS 167
- DB-STORE 178
- IDM-CONNECT 283, 284

RECORD CONTAINS Report Writer clause 74

RECORD keyword

- MSG-SW 339, 433
- NTRY 343

RED keyword 378, 404

REDEF keyword

- SCRNLIST 427

REDEFINES clause in data structures 379

redefining

- data elements 379
- screen records 424, 426

REF keyword

- DB-MODIFY 140, 141
- DB-OBTAIN 151, 153
- DB-PROCESS 167

DB-STORE 178

REFERENCE Report Writer clause 380, 405 registers

- see special registers, SQL

relational operators in S-COBOL 417

relative byte address

- VSAM online 261

relative record number

- VSAM batch 259
- VSAM online 261

release

- see CIC-RELEASE call

REM keyword 382

RENAMES clause in data structures 13

REPEAT LINKING statement 383

REPEAT statement 264, 383

REPEAT VARYING statement 383

REPLACE

- see DLG-VDEFINE call

report file description keyword 247

REPORT FOOTING (RF) keyword 479

Report Generator

- summary of reports 32, 407

REPORT HEADING (RH) keyword 479

report mock-ups

- data fields 388
- identifying in Report Section 333
- limits 320
- line limits 334
- literal fields 388, 491
- naming conventions 25
- painting 388
- PIC string 388
- record size 74
- specifying in application definition 25

Report Section

- keywords 11, 333, 378
- Report Writer 11, 333, 378

Report Writer

- 01 keyword 11
- accumulators 305
- accumulators, sum 380, 438, 447
- begin processing 305
- coding rules 405

- coding your own WRITE statement 495
- control breaks 78, 269, 379
- counters 305
- declaratives 485
- defining the report 378
- detail lines 475
- detail reports 269, 475
- end processing 461
- FD keyword 247
- File Section 247
- footers 475
- headers 475
- identifying mock-up 333
- initializing accumulators 305
- Input-Output Section 305
- iterative expressions 438, 447
- keywords and structures 404
- large reports 270
- limits 353
- literal values 491
- mapping data items 438
- MOCK keyword 333
- multiple detail lines 447
- non-printing fields 380
- overview 404
- Procedure Division statements 269, 305, 461, 485
- processing each report 269
- record length 74
- RED keyword 378
- report group types 475
- Report Section 378
- sample report programs 389
- summary reports 269, 475
- summing data items 380, 438, 447
- suppressing printing 450
- USE BEFORE REPORTING 485
- reports, APS
 - Application Definition (AP01) 16, 407
 - Component List (MS01) 76, 407
 - Data Structure Definition (DS01) 107, 407
 - DDIFILE (DB01) 181, 407
 - Entity Content (MS02) 210, 407
 - Entity Cross Reference (MD01) 212, 407
 - Entity Parts List (EN01) 214, 407
 - Entity Search Utility (GS01) 216, 407
 - Entity Use (EN02) 219, 407
 - Field Edits (ED01) 407, 420
 - Field/Screen Cross Reference (SC02) 262, 407
 - generating 32, 407
 - Macro/Program Cross Reference (MC01) 327, 407
 - Mock-Up (RP01) 335, 407
 - Program DB/DC (PG02) 372, 407
 - Program Definition (PG01) 374, 407
 - Scenario Definition (CN01) 407, 419
 - Screen Hardcopy/Field Attribute (SC01) 407, 420
 - summary of 32, 407
- REQID parameter
 - CIC-CANCEL 56
 - CIC-DELAY 57
- reserved words
 - COBOL and S-COBOL 413
 - complete listing 409
- RESET
 - see DLG-VRESET call
- RESET keyword
 - DB-OBTAIN 152
 - DB-PROCESS 168, 171
- RESET Report Writer clause 447
- RESETBR keyword
 - DB-OBTAIN 152
- RESET-OBTAIN flag
 - VSAM batch 258
 - VSAM online 259
- RESET-PFKEY call 411
- restart
 - see checkpoint/restart
- RETRY keyword
 - NTRY 343
- RETRY parameter
 - NTRY 83
- RETURN 287
- RETURN keyword
 - NTRY 343

REWRITE parameter
 CIC-WRITEQ-TS 70
 rewriting IDMS DB records 138
 RI-ON-REC flag
 SQL 232
 ROLLBACK
 see DB-ROLLBACK call, IDM-ROLLBACK call
 RRDS support
 VSAM batch 259
 VSAM online 261
 RTY-ON-REC flag
 IMS DB 227
 ruled line attribute, KANJI format 45
 RUN-UNIT keyword
 DB-BIND 119

S

SAGE-TRACE-FLAG 233
 scalar functions
 see functions, SQL
 Scenario Definition Report (CN01) 407, 419
 Scenario Painter
 field limit 320
 SCHEDULE-PSB
 see CIC-SCHEDULE-PSB call
 S-COBOL
 abbreviated syntax 417
 coding rules 415
 comments 75
 continuation 416
 differences from COBOL 413
 flags 415
 indentation 357, 413
 limits 321
 overview 413
 punctuation 416
 verbs 414
 S-COBOL structures 413
 screen fields
 generation parameters 273
 generation parameters, for all targets 273
 generation parameters, for IMS 329, 330, 332
 generation parameters, for ISPF Dialog 355
 generation parameters, for KANJI format 274
 length, changing 43
 limits 320
 LOW-VALUES 471
 MFS, system literals, defining 45
 MFS, trancodes, creating 330
 MFS, trancodes, literal values 278
 MID MOD reordering 332
 naming conventions 42
 null values 471
 value, initial 43
 Screen Hardcopy/Field Attribute Report (SC01) 407, 420
 SCREEN keyword
 MSG-SW 339
 SEND 433
 Screen Painter
 field limits 320
 SCREEN-INVOKED
 see TP-SCREEN-INVOKED flag
 screens
 attributes, overriding 40
 attributes, resetting 40, 71, 471
 displaying 340, 431
 field edits, performing 340
 generating in Linkage Section 426
 generation parameters 273
 generation parameters, for all targets 273
 generation parameters, for IMS 329, 330, 332
 generation parameters, for ISPF Dialog 355
 generation parameters, for KANJI format 274
 I/O areas, reporting on 372
 LOW-VALUES 71

- naming conventions 25
- null values 71
- receiving multiple screens 426
- redefining 424, 426
- sending messages 431
- sending multiple pages 435
- sending single or multiple screens 431
- simulating screen invocation 411
- size, specifying 25
- specifying in application definition 25
- specifying in NTRY 340
- screens, APS
 - Application Selection 19
 - Bind Options 46
 - Checkin, ENDEVOR Interface 51
 - Checkout, ENDEVOR Interface 53
 - Field Attributes 42
 - Generation Options 271
 - IDMS options 293
 - ISPF Panel Options 355
 - Job Control Cards 307
 - MFS Function Keys 329
 - MFS Trancode Construction 330
 - MID MOD Reorder 332
 - Parm 240
 - Precompiler Options 364
 - Project Group Environment 376
- SCRNLIST call 426
- SD keyword 429
- SEARCH statement 430
- SEC-VIO flag
 - IMS DC 230
- SEG-NOT-FOUND flag
 - IMS DC 230
- SELECT statement keyword 305
- SELECTIVE keyword
 - DB-ERASE 132
- SEND call 431
- SEND-TEXT
 - see CIC-SEND-TEXT call
- SEQUENCE keyword
 - DB-OBTAIN 152
- SERVICE-RELOAD
 - see CIC-SERVICE-RELOAD call
- SET ERROR Customization Facility statement
 - 81
- SET keyword
 - DB-OBTAIN 152
 - IDM-IF 285
- SET parameter
 - CIC-GETMAIN 60
 - CIC-LOAD 62
 - CIC-READQ-TD 63
 - CIC-READQ-TS 64
- SETMSG
 - see DLG-SETMSG
- SHORT keyword
 - DLG-SETMSG 202
- size limitations in APS 319
- software library 26
- Sort Description
 - keyword 429
- SORT procedure keyword 357
- SOURCE Report Writer clause 405, 438, 447
- SPA (Scratch Pad Area) 465
- SPA-IO-ERR flag
 - IMS DC 230
- special registers, SQL 440
 - DB-DECLARE 440
 - DB-OBTAIN 440
 - DB-PROCESS 440
- Special-Names keyword 441
- Specification Editor
 - DB calls 97
 - DC calls 101
- SPNM keyword 441
- SQL
 - abnormal condition processing 233
 - adding rows 175, 176
 - APDB2IN control file 85
 - bind and translate generation options 46
 - calls, list of 98
 - closing cursor sets 120, 160, 161
 - committing 121
 - copybooks, customize 194
 - copybooks, placement 83
 - cursor naming 125, 165
 - customization exits 133, 172

- declaring table rows 122, 160, 161
- deleting rows 128
- error handling 232
- exit points 244
- expressions 246, 265
- functions 246, 265
- grouping data elements 278
- IMS considerations 371
- joining tables 307
- looping 160, 161
- modifying rows 138
- native (pass-through) support 441
- null indicators 349
- opening cursor sets 158, 160, 161
- Procedure Division statement 442
- processing rows 160, 161
- reading rows 134
- retrieving rows 134
- rollback functions 174
- selecting rows 143, 147, 160
- special registers 440
- specifying as target 24, 118
- status flags 232
- storing rows 175, 176
- subschemas 180
- topics, list of 100
- unions 480
- writing rows 175, 176
- SQL keyword 441
- SQL Server
 - exit points 244
- starting
 - see CIC-START call
- STOP RUN statement 444
- STORE
 - see DB-STORE call
- STUB keyword 444
- stubs
 - see global stubs or local stubs
- subroutines/subprograms
 - calling 321
 - invoking with ENTRY statement 221
 - invoking with LINK call 321
 - invoking with XCTL call 497
- SUBSCHEMA keyword
 - DB-BIND 119
- subschemas
 - DB-SUBSCHEMA call 180
 - in Program DB/DC Report 372, 407
 - naming conventions 26
 - specifying in application definition 26, 180
- SUBSCRIPT keyword
 - DB-ERASE 132
 - DB-OBTAIN 152
 - DB-PROCESS 168
 - DB-STORE 178
- sum accumulators, Report Writer 380, 438, 447
- SUM function 246
 - SQL 265
- SUM Report Writer clause 380, 405, 447
- summary reports, Report Writer 269, 475
- SUPPRESS option 450
- SUPPRESS Report Writer statement 450
- SUPRA statements 451
- SY keywords 452
- SY macro keywords 465
- syntax, abbreviated for S-COBOL 417
- SYSID keyword
 - DB-OBTAIN 152
- SYSID parameter
 - CIC-CANCEL 57
 - CIC-DELETEQ-TD 58
 - CIC-DELETEQ-TS 59
 - CIC-READQ-TD 63
 - CIC-READQ-TS 64
 - CIC-WRITEQ-TD 69
 - CIC-WRITEQ-TS 70
- SYSIDERR error flag/condition 236
- SYSMSG field
 - creating 274
- system messages
 - creating field for 274
- system service calls
 - IMS Fast Path 454

T

- task processing, CICS
 - starting 67
 - stopping 57
- teleprocessing functions, Online Express
 - customizing 88
- temporary storage area, CICS 59, 63, 70
- TERM call 460
- TERMINATE Report Writer statement 461
- terminating programs 245, 444, 460, 462
- TERM-PSB
 - see CIC-TERM-PSB call
- time field edits
 - input 117
 - internal picture 116
 - output 117
- TIME function, SQL 265
- TIME parameter
 - CIC-DELAY 57
 - CIC-START 68
- TIME special register, SQL 440
- TIMESTAMP special register, SQL 440
- TIMEZONE special register, SQL 440
- TO keyword 283, 284
- TP-ATTR
 - see ATTR call
- TP-BACKOUT call 462
- TP-CLEAR
 - see CLEAR call
- TP-CLEAR-ATTRS
 - see CLEAR-ATTRS call
- TP-COMMAREA call
 - CICS 323, 463, 498
 - CICS, passing 323, 463
 - IMS DC 465
 - ISPF Dialog 466
 - ISPF Dialog, passing 323, 467
- TP-ENTRY
 - see NTRY keyword
- TP-LINK
 - see LINK call
- TP-LINKAGE call 468
- TP-LINK-INVOKED flag 258
- TP-MSG-SW MSG-SW call see 337
- TP-NULL call 471
- TP-PERFORM call 472
- TP-PGM-ERR flag
 - IMS DC 230
- TP-PROGRAM-INVOKED flag
 - CICS 256
 - IMS DC 257
 - ISPF Dialog 258
- TP-RESET-PFKEY RESET-PFKEY call see 411
- TP-SCREEN-INVOKED flag
 - CICS 256
 - IMS DC 257
 - ISPF Dialog 258
- TP-SCRNLIST SCRNLIST call see 426
- TP-SCRN-RECEIVED flag 427
- TP-SEND SEND call see 431
- TP-TERM TERM call see 460
- TP-TRANSID-INVOKED flag
 - CICS 256
 - IMS DC 257
 - ISPF Dialog 258
- TP-USERAREA 49, 498
 - CICS 463
 - IMS DC 465
 - ISPF Dialog 466
- TP-USER-LEN 49
 - CICS 463
 - IMS DC 465
 - ISPF Dialog 466
- TP-XCTLXCTL call see 497
- trace facility
 - SCBTRACE 364
- trancodes
 - creating 330
 - literal values, specifying 278
- transaction ID, specifying 275
- transferring to other programs 321, 497
- TRANSID keyword
 - SEND 434
- TRANSID parameter
 - CIC-CANCEL 57
 - CIC-START 68

TRANSID-INVOKED TP-TRANSID-INVOKED
 flag see 256
 Transid-Invoked-Para control point 89
 transient data, CICS 58, 62, 69
 translate options, SQL 46
 TRUE flag 475
 TRUE statement 473
 Ty(pe) field, Application Painter 25
 TYPE Report Writer clause 405, 475

U

unions
 DB-DECLARE 480
 DB-PROCESS 480
 UNLOCK keyword
 DB-FREE 136, 172
 unprotected fields 274
 UNTIL statement 383, 483
 UPDATE keyword
 DB-DECLARE 126
 DB-PROCESS 168
 UPON Report Writer clause 447
 USAGE clause in data structures 110
 Use 55
 USE BEFORE REPORTING Report Writer clause
 485
 USE BEFORE REPORTING Report Writer state-
 ment 405
 user exits
 Application Painter 27
 user help 486
 APHPIN control file 88
 user-defined field edits
 control file switches 84
 creating 18
 examples 22
 referencing APS-generated field names
 in 19
 selecting from predefined list 19
 USERMACs
 application limit 320

USERNAME statement 489, 490
 USING keyword 287

V

VALUE clause in data structures 14, 490
 value ranges, field edits 493
 VALUE Report Writer clause 405, 491
 value, screen field, initial 43
 variable length file support
 VSAM batch 494
 VSAM online 262
 VCOPY DLG-VCOPY call see 203
 VDEFINE DLG-VDEFINE call see 204
 VDELETE DLG-VDELETE call see 205
 VIEW keyword
 DB-ERASE 131, 132
 DB-FREE 136
 DB-MODIFY 140, 141
 DB-OBTAIN 151, 152, 153
 DB-PROCESS 167, 168
 DB-STORE 177
 VIEW parameter
 DB-STORE 179
 VIO-ON-REC flag
 IDMS DB 225
 IMS DB 227
 VREPLACE DLG-VREPLACE call see 206
 VRESET DLG-VRESET call see 207
 VSAM
 specifying as target 24, 118
 VSAM batch
 abnormal condition processing 235
 APVSAMIN control file 87
 closing files 120
 customization exits 134
 DDI statements 182
 deleting records 128
 enable prototype mode 82
 error handling 234
 ESDS support 259
 exit points 244

- fields and flags 234, 258
- modifying records 138
- opening 158
- reading records 143, 147
- retrieving and processing records 160, 163
- RRDS support 259
- storing records 175, 176
- subschemas 180
- topics, list of 100
- variable length file support 494
- writing records 175, 176

VSAM online

- abnormal condition processing 237
- APVSAMINcontrol file 87
- calls, list of 99
- customization exits 134
- DDI statements 182
- deleting rows 128
- enable prototype mode 82
- error handling 236
- ESDS support 261
- exit points 244
- fields and flags 236, 259
- modifying records 138
- reading records 143, 148
- releasing file resources 135
- retrieving and processing records 160, 163
- RRDS support 261
- storing records 175, 176
- subschemas 180
- topics, list of 100
- variable length file support 262
- writing records 175, 176

- WHERE keyword
 - DB-DECLARE 126
 - DB-ERASE 132
 - DB-MODIFY 141
 - DB-OBTAIN 152
 - DB-PROCESS 168
 - DB-STORE 178
- WHILE statement 383, 483
- WITH HOLD keyword
 - DB-DECLARE 126
 - DB-PROCESS 169, 172
- Working-Storage Section keyword 496
- WRITE ROUTINE Report Writer clause 248, 404, 495
- WRITEQ-TD CIC-WRITEQ-TD call see 69
- WRITEQ-TS CIC-WRITEQ-TS call see 70
- WS keyword 496

X

- XCTL call 497
 - CICS PSB 375
 - Commarea, passing 323
 - passing the Commarea 323, 463, 467
- XCTL call, Specification Painter
 - passing the Commarea 467

W

- WHERE CURRENT keyword
 - DB-ERASE 132
 - DB-MODIFY 141