



MICRO FOCUS

APS FOR z/OS

**CUSTOMIZATION FACILITY
USER'S GUIDE**



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

1	Customization Facility Concepts	5
2	Structures	15
3	Sample Macros	55
	% IF Structure Example	55
	Discussion	58
	Looping Example	59
	Discussion	61
	Customization Facility Functions Example	62
	Discussion	65
	The \$DAYDEF Macro	65
	The Program:	66
	Program Location Statements Example	66
	Discussion	69
4	Administration	73

1 Customization Facility Concepts

Use facility to modify APS macros and write your own

The APS Customization Facility is a high-level tool that lets you customize APS. With this tool, you can write macros--sometimes referred to as rules--to modify and supplement the default APS processing logic. Specifically, you can:

- Modify and extend the APS software macros, which reside in the APSMACS PDS or data set in the APS software.
- Write your own supplemental macros in the USERMACS PDS or data set in your APS Project and Group.

A Customization Facility macro is a source code component that you can reuse in any APS application. You write macros using high-level Customization Facility structures that are similar to COBOL structures, but offer more functionality with less coding. To call macros in your programs, simply assign macro calls to Online Express program functions and control points; when writing programs using the Program Painter, you simply write macro calls in the programs. In these calls, you pass values to variables in the macros according to your program requirements. For example, APS might process a macro that performs a loop and builds a table, based on the size, data name, and PIC clause values that you pass to it. You generate programs that invoke macros just as you generate any APS program. At generation, the Customization Facility processes the macros and includes them in your program; no extra generation steps are required.

The Customization Facility Control System lets APS administrators regulate developer access to user-defined Customization Facility macros. Use it to do any of the following:

- Restrict all developers from using any user-defined macros (default).
- Allow all developers to use any user-defined macro.
- Specify selected user-defined macros that all developers can use.

Types of source code

You can write macros containing any combination of the following types of source code:

- Customization Facility structures.
- COBOL or COBOL/2.
- S-COBOL, a set of COBOL-like APS programming structures that you can use in macros and APS programs. For information, see the APS Reference.

Customization Facility structures

The Customization Facility provides the following structures that you can use in macros and APS programs as well.

Structure	Description
<code>% BEGIN</code>	Positions source code in a specific column in the output.
<code>% DECLARE</code>	Lets you organize macro symbols into tables, providing an associative memory capability
<code>% DEFINE</code>	Use to begin defining a macro; assign its name and variables that receive values from macro calls.
<code>% ESCAPE</code>	Exits a macro.
<code>&function</code>	Executes predefined Customization Facility functions.
<code>% IF ...</code>	Evaluates one or more conditions.
<code>% ELSE-IF ...</code>	
<code>% ELSE</code>	
<code>% INCLUDE</code>	Opens, reads, and processes a file in a macro.
<code>% LOOKUP</code>	References a % DECLARE table.
<code>% REPEAT</code>	Establishes a loop.
<code>% SET</code>	Specifyies the program location where the Customization Facility places source code.
<code>% UNTIL/WHILE</code>	Establishes a loop and test a condition; use to add a condition test to a % REPEAT loop.
<code>% &variable = value</code>	Assigns a value to a variable.

Structure	Description
<>	APS-supplied macros are assigned "less-than" (<) and "greater than" (>) as evaluation brackets.

Customization Facility structures use the reserved symbols described below.

Symbol	Description
%	When % is the first character on a line, it identifies the entire line as a Customization Facility statement. When % is the first character in a pair of evaluation brackets, it identifies the source in the brackets as a Customization Facility statement.
&	<i>&dataname</i> is a variable. Generally, you code variables in macros and assign values to them in your APS programs. In addition, you can both define and assign values to variables in your programs. You can use variables in a Customization Facility statement or embed them in a line of COBOL or S-COBOL. <i>&functionname</i> is a predefined Customization Facility function.
\$	<i>\$macroname</i> is a macro definition statement or macro call. Generally, you code macro definitions in your USERMACS files, and macro calls in your programs. You can use macro calls in a Customization Facility statement or embed them in a line of COBOL or S-COBOL.
+	<i>&dataname+suffix</i> appends a literal suffix to a variable. <i>&columnnumber+source</i> designates the column where the facility processor places source code in the output.

Define macros

You define macros in the USERMACS PDS or data set in your APS Project and Group. All macros:

- Begin with a % DEFINE statement that assigns a name to the macro and can include any number of formal argument variables. A formal argument variable receives values, called actual arguments, from the macro calls that invoke the macro.

- Contain any combination of Customization Facility structures, COBOL or COBOL/2 source, or S-COBOL source.
- End with an % END statement, or any source code starting in the same column as the % DEFINE statement.

Include macros in applications

To include a macro in your application, you specify its macro library name on the Application Painter, in the User Mac field. In addition, you specify in the Location field the program location where you want to include it. Alternatively, you can code an % INCLUDE statement in the program at the desired location.

Invoke macros in programs

To invoke a macro, use one of the following methods depending on which APS tool you use to define your program:

APS Tool	Invocation Method
Online Express	Assign a macro call to a program function or control point, using the Alternate Functions, PF Key Functions, Special Key Definition, Control Points, or Database Call Tailoring window. In the call, you can pass values to the variables in the macro.
Program Painter	Write a macro call at the desired program location. In the call, you can pass values to the variables in the macro.

Generation processes

When you generate an application that invokes Customization Facility macros, APS processes the application as follows:

- Ensures that each component of your application exists.
- Generates APS symbols for the windows or screens, for use by the APS Precompiler.
- Generates the window or screen source native to your DC environment, such as BMS or MFS mapsets.

- Arranges all APS program specifications into proper COBOL program organization.
- Inserts all externally-defined components--such as APS macros, user-defined Customization Facility macros, COBOL copylibs, and APS data structures--at the program locations you specify in the programs or Application Painter.
- Processes all APS macro and user-defined Customization Facility macro symbols and structures into COBOL source as follows:
 - Replaces all variables with their assigned values
 - Passes all actual arguments of macro calls to the formal arguments of their corresponding macro definitions
 - Builds loop tables and inserts them in the program
 - Replaces all Customization Facility functions with their underlying source code
 - Moves all source code to the program locations you specify using % SET statements
 - Calls all macros and files called within other macros and processes them
- Processes all APS database and data communications calls into COBOL source.
- Translates any APS Report Writer source to COBOL source.
- Writes a temporary error message file and sorts it into the COBOL compiler error message file. The combined error message file presents messages sorted by program line number, with both types of messages appearing where appropriate.

Macro processing example

The following example shows two sample user macros--\$CONSTRUCT-FD and \$DAYDEF--and how a program invokes and uses them. To illustrate how APS processes macros, we show the program at two stages of generation:

- Before APS inserts the macro source in the program. The program at this stage is stored in the GENSRP PDS or data set in your Project and Group.

- After APS inserts the macro source in the executable COBOL program.

The macros are defined in a USERMACS file as shown below. The \$CONSTRUCT-FD macro writes FD statements. The developer calls the macro in the program and passes arguments--the FD file names--to the FD file name variable in the macro. The \$DAYDEF macro builds a Working-Storage symbol table. The developer calls the macro repeatedly in the program and passes arguments--literal strings and their lengths--to the string and length variables in the macro.

User-defined macros:

```
% DEFINE $CONSTRUCT-FD( &FILE-NAME)
    FD &FILE-NAME
        BLOCK CONTAINS 0 RECORDS
        LABEL RECORDS ARE STANDARD.
% END

% DEFINE $DAYDEF( &DAY, &LEN)
    % &DAYCTR = &DAYCTR + 1
    02 FILLER.
        03 FILLER                PIC S9(2) COMP SYNC VALUE +&LEN.
        03 FILLER                PIC X(9) VALUE &QT&DAY&QT.
% END
```

Program before APS includes the macro source:

```
% &REC-LEN = 121                                315.
% &DAYCTR = 0                                    316.
% &QT = " ' "                                    317.
                                                    318.
IDENTIFICATION DIVISION.                        319.
PROGRAM-ID.                                     EXAMPLE3. 320.
*SPECIAL CONSIDERATIONS.                       321.
*   SAMPLE PROGRAM: CALLS TO MACROS THAT WRITE 322.
*   AN FD STATEMENT AND BUILD A TABLE.        323.
                                                    324.
ENVIRONMENT DIVISION.                           325.
INPUT-OUTPUT SECTION.                          326.
FILE-CONTROL.                                   327.
    SELECT INPUT-FILE                           ASSIGN-UT-S-INPUT. 328.
    SELECT OUTPUT-FILE                          ASSIGN-UT-S-OUTPUT. 329.
                                                    330.
DATA DIVISION.                                  331.
```

FILE SECTION.	332.
	333.
\$CONSTRUCT-FD('INPUT-FILE')	334.
01 INPUT-RECORD PIC X(&REC-LEN).	335.
	336.
\$CONSTRUCT-FD('OUTPUT-FILE')	337.
01 OUTPUT-RECORD PIC X(&REC-LEN).	338.
WORKING-STORAGE SECTION.	340.
	341.
01 DAY-TABLE.	342.
\$DAYDEF('SUNDAY', 6)	343.
\$DAYDEF('MONDAY', 6)	344.
\$DAYDEF('TUESDAY', 7)	345.
\$DAYDEF('WEDNESDAY', 9)	346.
\$DAYDEF('THURSDAY', 8)	347.
\$DAYDEF('FRIDAY', 6)	348.
\$DAYDEF('SATURDAY', 8)	349.
01 DAY-TABLE-REDEF REDEFINES DAY-TABLE	350.
02 DAY-ENTRY OCCURS &DAYCTR.	351.
03 DAY-LEN PIC S9(2) COMP SYNC.	352.
03 DAY-BOL PIC X(9).	353.

Program after APS includes the macro source:

030200	315
030800	316.
031100	317.
031800	318.
031900 IDENTIFICATION DIVISION.	319.
032000 PROGRAM-ID. EXAMPLE3.	320.
032100*SPECIAL CONSIDERATIONS.	321.
032200* SAMPLE PROGRAM: CALLS TO MACROS THAT WRITE	322.
032300* AN FD STATEMENT AND BUILD A TABLE	323.
032400	324.
032500 ENVIRONMENT DIVISION.	325.
032600 INPUT-OUTPUT SECTION.	326.
032700 FILE-CONTROL.	327.
032800 SELECT INPUT-FILE ASSIGN-UT-S-INPUT.	328.
032900 SELECT OUTPUT-FILE ASSIGN-UT-S-OUTPUT.	329.
033000	330.
033100 DATA DIVISION.	331.
033200 FILE SECTION.	332.
033300	333.
033400 FD INPUT-FILE	334.
033402 BLOCK CONTAINS 0 RECORDS	334.
033404 LABEL RECORDS ARE STANDARD.	334.
033500 01 INPUT-RECORD PIC X(121).	335.

033600				336.
033700	FD	OUTPUT-FILE		337.
033702		BLOCK CONTAINS 0 RECORDS		337.
033704		LABEL RECORDS ARE STANDARD.		337.
033800	01	OUTPUT-RECORD	PIC X(121).	338.
033900				339.
034000		WORKING-STORAGE SECTION.		340.
034100				341.
034200	01	DAY-TABLE.		342.
034300	02	FILLER.		343.
034302	03	FILLER	PIC S9(2) COMP SYNC VALUE +6.	343.
034304	03	FILLER	PIC X(9) VALUE 'SUNDAY'.	343.
034400	02	FILLER.		344.
034402	03	FILLER	PIC S9(2) COMP SYNC VALUE +6.	344.
034404	03	FILLER	PIC X(9) VALUE 'MONDAY'.	344.
034500	02	FILLER.		345.
034502	03	FILLER	PIC S9(2) COMP SYNC VALUE +7.	345.
034504	03	FILLER	PIC X(9) VALUE 'TUESDAY'.	345.
034600	02	FILLER.		346.
034602	03	FILLER	PIC S9(2) COMP SYNC VALUE +9.	346.
034604	03	FILLER	PIC X(9) VALUE 'WEDNESDAY'.	346.
034700	02	FILLER.		347.
034702	03	FILLER	PIC S9(2) COMP SYNC VALUE +8.	347.
034704	03	FILLER	PIC X(9) VALUE 'THURSDAY'.	347.
034800	02	FILLER.		348.
034802	03	FILLER	PIC S9(2) COMP SYNC VALUE +6.	348.
034804	03	FILLER	PIC X(9) VALUE 'FRIDAY'.	348.
034900	02	FILLER.		349.
034902	03	FILLER	PIC S9(2) COMP SYNC VALUE +8.	349.
034904	03	FILLER	PIC X(9)VALUE 'SATURDAY'.	349.
035000	01	DAY-TABLE-REDEF	REDEFINES DAY-TABLE.	349.
035100	02	DAY-ENTRY	OCCURS 7.	351.
035200	03	DAY-LEN	PIC S9(2) COMP SYNC.	352.
035300	03	DAY-BOL	PIC X(9).	353.

Related Topics

See...

% BEGIN

Functions, Customization Facility

% DECLARE

For more information about...

Positioning source code in a specific column in the output

Organizing macro symbols into tables, providing an associative memory capability

See...	For more information about...
<i>% DEFINE</i>	Writing macro definition statements
<i>% ESCAPE</i>	Exiting macros
<i>Functions, Customization Facility</i>	Executing predefined Customization Facility processes
<i>% IF/ELSE-IF/ELSE</i>	Evaluating conditions
<i>Evaluation Brackets</i>	Specifying the order in which the Customization Facility evaluates variable values
<i>% INCLUDE</i>	Opening, reading, and processing files in macros
<i>% LOOKUP</i>	Referencing <i>% DECLARE</i> tables
<i>% REPEAT</i>	Establishing loops and testing conditions
<i>% UNTIL/WHILE</i>	
<i>% SET Statements</i>	Specifying the program location where the Customization Facility places source code; overriding processing defaults
<i>Variable Assignment Statements</i>	Assigning values to variables

2 Structures

This chapter lists the structures you can use to customize APS.

% BEGIN

Description: Use in conditional or looping statement blocks to control the number of columns that APS moves the blocks during processing. The number of columns that % BEGIN is indented from a conditional or looping construct is the same number of columns that APS moves the subordinate block to the left. For example, if % BEGIN is indented four columns from an IF statement, all lines subordinate to % BEGIN move four columns to the left during processing. Any text at the same or lesser indentation than % BEGIN stops the effect of % BEGIN, and default processing resumes. Default processing moves the block to the starting column of the controlling conditional or looping construct.

Syntax: % BEGIN

Comments: Do not use the Customization Facility function *&columnnumber+source* with % BEGIN.

Examples: Override default processing to shift the IF statement's subordinate statement block to start at column 12. Default processing shifts the block instead to the starting column of the IF statement--column 8--as shown below:

Input:

```
Column:  8...12..16
          % IF &TYPE = 'NEW'
            01  INV-ITEM.
              05  PART-CLASS.
                10  PART-NUMBER PIC X(5).
```

Default output:

```
Column:  8...12..16
         01  INV-ITEM.
           05  PART-CLASS.
           10  PART-NUMBER PIC X(5).
```

Use % BEGIN to force the statement block to start at column 12:

Input:

```
Column:  8...12..16
         % IF &TYPE = 'NEW'
           % BEGIN
           01  INV-ITEM.
             05  PART-CLASS.
             10  PART-NUMBER PIC X(5).
```

Output:

```
Column:  8...12..16
         01  INV-ITEM.
           05  PART-CLASS.
           10  PART-NUMBER PIC X(5).
```

Related Topics:	See...	For more information about...
	<i>Functions,</i> <i>Customization Facility</i>	Specifying the column number to place source code in the output
	% REPEAT	Placement of looping structure output
	% UNTIL/WHILE	

Comments

Description: Document Customization Facility source code with Comments.

Syntax: %* *comment text*

Comments:

- A comment can start in any column, but must appear on a line by itself.

Continue a macro call.

```
$TABLE-MAKER( 3, 4, 'ENTRY', 'ITEM',
... 'X(4) VALUE SPACES')
```

Continue a literal string variable value by coding a hyphen anywhere in the string.

```
% &VARIABLE = 'THIS WORKS-
% ... FINE'
% &VARIABLE = 'THIS WORKS FI-
% ... NE TOO'
```

Continue other statements where a space occurs.

```
% REPEAT VARYING &IM-TARG FROM -
% ... 1 TO 20
% IF &APS-IDENT < 8 -
% ... OR &APS-INDENT > 11
```

Related Topics: See...

APS Reference:
 "About Data Communication Calls"
 "About Data Structures"
 "About Database Calls"
 "About S-COBOL Structures"

For more information about...

Continuing APS statements

% DECLARE

Description: Define a % DECLARE table made of variable assignments that can be referenced either directly (after assigning values to its subscripts), or by searching with the % LOOKUP structure. Help access a loaded table made of variable assignments.

Syntax:

```
% DECLARE &fieldname(&subscript1)[...(&subscriptN)]
[ % &declarepart1 [Xn|Nn|REDEFINES]
.
.
.
% &declarepartN [Xn|Nn|REDEFINES]]
[% END]
```

- Parameters:** *&declarepart* Subscript-part of *&fieldname*. Indentation indicates subordination. Note the following:
- Do not use an *&declarepart* as a *&subscript*.
 - If an alphanumeric or numeric picture clause (Xn or Nn) assigns a fixed length to a *&declarepart*, and *&fieldname* has more than one *&declarepart*, each *&declarepart* needs a picture clause.
 - The maximum *&declareparts* sum in one DECLARE structure level is 78 bytes.
 - The maximum number of *&declareparts* per table is 1000.
- &fieldname* Name-assignment statement with zero or more subscripts and up to 78 *&declareparts* in each structure level.
- &subscript* Set members of single- or multi-dimensional arrays. Values are numbers or strings (maximum 12 characters). For numeric values, the counter starts at 1 and increments until the end of the numeric value. The maximum number of *&subscripts* per program is 300.
- Comments:**
- Assign at least one *&declarepart* of each *&fieldname* a value in order to reference other *&declareparts* for the same *&fieldname*.
 - Values of *&declareparts* can be changed by conventional APS variable assignment statements.
 - *&Declareparts* can be accessed in the same manner as any other variable except when accessed by the following functions: *&INDEX*, *&LENGTH*, *&NUMERIC*, *&PARSE*, *&SUBSTR*.
 - *&Declareparts* can be redefined with REDEFINES or REDEF. REDEFINES indicates that it shares the same DECLARE storage as the *&declarepart* immediately preceding it at the same level of indentation.
 - The maximum number of % DECLARE tables per program is 200.
- Examples:**
- Although *&subscript* *&KEY* has already been set to several values, the most common value is 1. The current values of *&FILE* and

&KEY&FILE-KEY-NAME determine **&FILE-KEY-1-NAME**, whereas **&FILE-KEY-1-NAME** always references key 1, regardless of the **&KEY** value.

```
% DECLARE &( &FILE )-KEY-( &KEY )-A
% &FILE-KEY-NAME          X30
% &FILE-KEY-TYPE          X
% &FILE-KEY-LEN           N4
% DECLARE &( &FILE )-KEY-1-A
% &FILE-KEY-1-NAME        X30
% &FILE-KEY-1-TYPE        X
% &FILE-KEY-1-LEN         N4
.
.
.
```

- Access fixed-length strings of screen field attribute records that are stored in the **SCRSYMB** file. The following **% DECLARE**:

```
% DECLARE &( &SCX-SCR )-XFLD-( &SCX-FLD )
% &SCP-FLD-NAME           X16
% &SCP-REP-CNT            N3
% &SCP-REP-COL            N3
% &SCP-FLD-SHORT          X8
% &SCP-FLD-FI
% &SCP-FLD-LEN            N3
% &SCP-FLD-ROW            N3
% &SCP-FLD-COL            N3
% &SCP-FLD-ATTRS
% &SCP-FLD-PROT           X
% &SCP-FLD-INTENS-FLG     X
% &SCP-FLD-MDT-FLG        X
% &SCP-FLD-NUM-FLG        X
% &SCP-FLD-EXT-ATTRS
% &SCP-FLD-RVID-FLG       X
% &SCP-FLD-BLINK-FLG     X
% &SCP-FLD-UNDER-FLG     X
% &SCP-FLD-COLOR          X2
% &SCP-FLD-DET-FLG        X
% &SCP-FLD-EDIT-FLG       N
% &SCP-FLD-IM-MOD         X
```

- Accommodates these fixed-length record strings stored in **SCRSYMB**:

```
% &DMOL-XFLD-1 = "PART-NBR   0 08   8   6 19UNTTTTFF  F1F"
% &DMOL-XFLD-2 = "SHORT-DESC 0 0   13   8 19PNTTTTTFF  F0F"
% &DMOL-XFLD-3 = "LOCATION     5 3   12  12 5PNTTTTTFF  F0F"
.
.
.
```

- Directly reference a DECLARE table.

```

% DECLARE &VS- (&VSX-FILE)-KEY-(&VSX-KEY)-B
% &VSP-FILE-KEY-VOL X8
% &VSP-FILE-KEY-SPACE X20
% &VSP-FILE-KEY-CICZ X20
.
.
.
% &VSX-FILE = "FILE3"
% &VSX-KEY = 2
.
.
.
% &VOL = &VSP-FILE-KEY-VOL

```

- Reference a DECLARE table with % LOOKUP.

```

% DECLARE &IMS-PCB- (&PCBX)-SEG- (&SEGX)
% &IMS-PCB-SEG-NAME X30
% &IMS-PCB-SEG-IMSNAME X8
% &IMS-PCB-SEG-PROCOPTS
% &IMS-PCB-SEG-PROCOPT-GET N
% &IMS-PCB-SEG-PROCOPT-ISRT N
% &IMS-PCB-SEG-PROCOPT-REPL N
% &IMS-PCB-SEG-PROCOPT-DLET N
% &IMS-PCB-SEG-LEN N6
.
.
.
% LOOKUP &IMS-PCB-SEG-NAME = &THE-SEG-NAME-YOU-WANT FROM 1 1
.
.
.
% ELSE
.
.
.

```

- Reference a loaded table by coding a table of value assignments; use % DECLARE to name the table and define &subs for the table name.

```

% DECLARE &OPERATOR- (&OPR)
% &OPERATOR X2
% &OPERATOR-SYMBOL X
% &OPERATOR-1 = "EQ="
% &OPERATOR-2 = "LT<"
% &OPERATOR-3 = "GT>"

```

```

% &OPERATOR-4 = "MI-"
% &OPERATOR-5 = "PL+"
% &OPERATOR-6 = "TIX"
% &OPERATOR-7 = "DI/"

```

- Load a table of value assignments by varying the *&subs* of a % DECLARE statement.

```

% DECLARE &VS-OPT-(&VSX-OPT)
% &VS-FILE-OPT
% REPEAT VARYING &VSX-OPT FROM 1
% WHILE &DEFINED(&SCR-OPTS-<&VSX-OPT>)
% &VS-FILE-OPT = &DEFVAL

```

Related Topics:	See...	For more information about...
	% LOOKUP	Referencing a % DECLARE table

% DEFINE

Description: Specify the name and formal arguments when defining a macro in a USERMACS file or APS program.

Syntax: % DEFINE \$macroname [(*&formalarg1* [, *&formalarg2*,
&formalarg3, ..., *&formalarg1000*])]
statementblock
[% END]

Parameters: *&formalarg* Formal argument list. Formal arguments receive their values from actual arguments in a macro call. Separate arguments with a comma and optionally one or more spaces. Enclose the argument list in parentheses.

\$macroname Valid COBOL name.

Examples: Define a macro with three formal arguments whose values are supplied by a call that invokes the macro.

```

% DEFINE $ITEM-MAKER( &MM, &DATANAME, &TAIL)
. . .
% END

```

Macro call:

```
$ITEM-MAKER( 12, 'TEST-ITEM', 'S9(9) COMP SYNC VALUE -1')
```

Related Topics:	See...	For more information about...
	<i>Macro Call</i>	Calling macros

Error Handling, Macros

Compatibility All targets

Description: APS writes error messages to a temporary file, passes them to the APS Precompiler, and displays them in the final APS message report of the compile.

Sending Messages

Send your own messages to the report, classified with a severity level, by coding one of the following SET statements:

```
% SET FATAL messagetext
% SET ERROR messagetext
% SET WARNING messagetext
% SET INFO messagetext
```

% SET FATAL ends all translation. Enclose variables in the message text within evaluation brackets.

Error Trace % SET TRACE ERROR is an error trace mechanism that appears by default in the APS CNTL file APSDBDC. The trace identifies:

- Line of source that caused the error
- Active % INCLUDE statements
- Active macros
- Number of loops completed at the time of error (if applicable)

The severity codes of errors traced are: F(atal), E(rror), W(arning), and I(nformation). To exclude Information messages, append the keyword NOINFO to the % SET TRACE ERROR statement. To trace a selected

section of your macro or program, code `% SET TRACE ERROR` and the beginning of the section, and `% SET NOTRACE` at the end.

Debugging with WRITE-CONTROL

`% SET WRITE-CONTROL` prints all Customization Facility input source in the output source, enabling you to view both the input and output source together, in context. It prints all error messages immediately below the source lines in error; they appear as COBOL Comments. This feature is especially helpful when examining complicated evaluation bracket processing.

Related Topics:	See...	For more information about...
	<i>% SET Statements</i>	Coding other <code>% SET</code> statements
	<i>Evaluation Brackets</i>	Processing with evaluation brackets

% ESCAPE

Description: Stop processing the remaining lines of a macro and resume processing with the line immediately following the macro call.

Syntax: `% ESCAPE`

Related Topics:	See...	For more information about...
	APS Reference: "ESCAPE" "STOP RUN"	Terminating APS programs
	APS Reference: "MSG-SW" "LINK" "XCTL"	Calling or linking to other programs or subprograms

Evaluation Brackets

Description: Specify the order in which the Customization Facility evaluates variable values.

Syntax:

Format 1: ... <% *source*> [...]

Format 2: ... <*source*> [...]

Parameters: *source*

Can include:

- An arithmetic expression; can include variables, numbers, and literals.
- Variables

- Comments:**
- In a line that contains source in evaluation brackets, the Customization Facility processor evaluates that source first.
 - In a line that contains multiple evaluation brackets, the processor evaluates the innermost bracket first, and the outermost bracket last.
 - The Customization Facility processor evaluates evaluation bracket source as follows:
 - Replaces Customization Facility variables with their assigned values
 - Calculates the result if the source is a non-Customization Facility expression--that is, if the first character inside the brackets is not the % character
 - Does not calculate the result if the source is a Customization Facility expression--that is, if the first character inside the brackets is the % character
 - Bracketed source in % UNTIL and % WHILE loops is evaluated at each iteration of the loop; source in % REPEAT and % REPEAT VARYING loops is evaluated just once.

- The default evaluation bracket characters are the < and > characters. Depending on how you use them, they can be either evaluation brackets or relational operators, as shown below.

Evaluation bracket:	<code><source></code>
Greater-than operator:	<code>source > source</code>
Less-than operator:	<code>source < source</code>
Greater-than or equal to operator:	<code>source >= source</code>
Less-than or equal to operator:	<code>source <= source</code>

- To override the default evaluation bracket characters, change the value of the % SET EVAL-BRACKETS statement in the APS CNTL file, APSDBDC. Do not define parentheses as bracket characters; we recommend using the square bracket characters [and] instead. You can also define a second, or auxiliary, set of evaluation brackets. When doing so, follow these rules:
 - Ensure that a first set of bracket characters is defined by the % SET EVAL-BRACKETS statement.
 - Define the auxiliary bracket characters by defining the % SET EVAL-BRACKETS-AUX statement:
- At intermediate stages in the use of evaluation brackets, a line of text can be expanded to exceed the right margin as long as the final line of output source fits in columns 7-72 (or 1-80 in Lang=Text mode, or 1-72 in Lang=JCL mode). Check the values on your APS Precompiler Options window.

Examples: Before Customization Facility processing:

```
% &VAR = 6
% &XVAR = 7
.
.
.
... <% &VAR + 3 + &XVAR>
.
.
... <&VAR + 3 + &XVAR>
```

After Customization Facility processing:

The Customization Facility processor replaces the variables in the first set of brackets with their values, but does not calculate the result because the source is a Customization Facility statement, as indicated by the % symbol inside the brackets. At the next processing step--APS generation--the APS generator calculates the result.

The Customization Facility processor replaces the variables in the second set of brackets with their values and calculates the result, because the source is not a Customization Facility statement.

```

.
.
.
... <6 + 3 + 7>
.
.
.
... <16>

```

Before Customization Facility processing:

```

% &ITEM-COUNT = 75
% &PKG-COUNT = <&ITEM-COUNT + 50> / 50

```

After Customization Facility processing:

The Customization Facility adds 75 to 50 and divides the result by 50; assigns the result--2--as the value of &PKG-COUNT.

Related Topics:

See...

% SET Statements
Macro Call

For more information about...

Defining evaluation bracket characters
Using evaluation brackets to code in-line macros

Functions, Customization Facility

Description: Perform predefined processes in a Customization Facility macro or APS program.

Syntax: &APS-EPILOGUE
 &APS-FULL
 &APS-HALF
 &APS-INDENT
 &APS-PROGRAM-ID
 &APS-PSB-NAME
 &columnnumber+source
 &COMPILETIME
 &DEFINED(\$macroname|&variablename)
 &DEFVAL
 &INDEX('wholestring', 'characterstring')
 &LENGTH(string)
 &NUMERIC(&variablename
 "literalstring"
 &PARSE(&string[, &variable])
 number
 &SUBSTR(string, startcolumn[, length])
 &dataname+-suffix

Description: &APS-EPILOGUE Returns the last macro call in the EPILOGUE queue. For information about calling macros with % SET EPILOGUE, see % *SET Statements*.

&APS-FULL Returns a full-word binary PIC, a string containing a 4-byte (full-word) binary picture specification: PIC S9(9) COMP.

&APS-HALF	Returns a half-word binary PIC, a string containing a 2-byte (half-word) binary picture specification: PIC S9(4) COMP.
&APS-INDENT	Returns the indentation level for the current macro. Contains the column position of the \$ of the invoking macro; otherwise, contains the column position of the left margin (usually column 7 for COBOL).
&APS-PROGRAM-ID	Returns the PROGRAM-ID name from the Identification Division.
&APS-PSB-NAME	Returns the user-specified PSB or subschema name for your program.
<i>&columnnumber+source</i>	Specifies the column number at which to place source code in the output.
&COMPILETIME	Returns the date and time of the compile: <i>dd mmm yy hh.mm.ss</i> .
&DEFINED	Determines whether a specific macro or variable has a defined value. Returns a true value (1) if defined, and a false value (0) if not defined.
&DEFVAL	Returns the string or number value of the last &DEFINED variable, the string or number in <i>&variablename</i> .
&INDEX	Searches a string left to right for the first occurrence of a character-string. Returns a number identifying its character position; return 0 if no character string. Delimiting quotes are not included in the character position. (See also &SUBSTR below.)
&LENGTH	Returns a number specifying the length of the argument string. Leading and trailing blanks are included as part of a string; delimiting quotation marks are not.
&NUMERIC	Determines whether a variable is numeric. Returns a true value (1) if numeric and a false value (0) if non-numeric.

&PARSE

Parses a Customization Facility text string. *&String* must be a variable with a string value. The second argument can be either a literal text string, a variable with a string value, or a number (the number turns into a string).

The second argument is matched against the first argument (the string value of *&string*). The part of the *&string* value following the match is stored back into *&string* with the leading and trailing spaces eliminated. Then, the part of the *&string* value preceding the match returns as the value of the parsing function, again with no leading and trailing spaces. In matching the second string inside the first string, the match does not begin within a string that is embedded (quoted) within the first string.

&SUBSTR

Extracts a substring from a string, starting with *startcolumn* and continuing for *substringlength*.

String can be an alphanumeric literal string delimited with quotation marks, or a variable with a string value.

Startcolumn and *substringlength* can be a number or variable; if *substringlength* is omitted, the substring includes all characters from *startcolumn* onward.

Note the following:

- If *startcolumn* is greater than *substringlength*, or if *substringlength* is 0, the result is an empty string.
- If *startcolumn* plus *substringlength* is greater than *substringlength*, the result begins with *string* start character and ends with its last character; the substring is not padded with blanks.

&dataname+suffix Appends a literal suffix to a variable data name by coding a plus symbol (+) in the space between them. During processing, the plus symbol disappears (it is not replaced with a space), and the suffix appends onto the resolved variable value.

Comments:

- Use the associative memory structures % DECLARE and % LOOKUP as alternatives to the following functions: &INDEX, &LENGTH, &NUMERIC, &PARSE, &SUBSTR.

Examples:

- Enforce the position of certain macros.

```
% DEFINE $TP-ENTRY
  % IF &APS-INDENT < 8 OR &APS-INDENT > 11
    % SET ERROR $TP-ENTRY MUST BE INVOKED
  % ... IN AREA A
```

- If &FILE-2 and &FILE-3 have defined values, add them to the close statement.

```
CLOSE &FILE-1
% IF &DEFINED( &FILE-2)
  ... &FILE-2
% IF &DEFINED( &FILE-3)
  ... &FILE-3
```

- Save two symbol lookups every time a loop is performed. Replace the following code:

```
% REPEAT VARYING &SC-I FROM 1
% WHILE &DEFINED( &<&SCR>-FLD-<&SC-I>)
  % &VAR = &<&SCR>-FLD-<&SC-I>
% END
```

- With this code:

```
% REPEAT VARYING &SC-I FROM 1
% WHILE &DEFINED( &<&SCR>-FLD-<&SC-I>)
  % &VAR= &DEFVAL
% END
```

- Search for the first occurrence of the character string is. The value 3 is returned.

```
&INDEX( 'THIS IS AN EXAMPLE', 'IS')
```

- Set &II to true and test for its numeric value.

```
% &II = 1
% IF &NUMERIC( &II )
    /* YOU WILL GET THIS TEXT
    .
    .
    .
% END
```

- Use + to append the suffix -ROUTINE to the variable &SCOPE and signify that they are two separate parts, not one variable called &SCOPE-ROUTINE.
- Input source:

```
% &SCOPE = 'CALC'
PERFORM &SCOPE+-ROUTINE
```

- Output source:

```
PERFORM CALC-ROUTINE
```

Extract a substring from the string value of &PREFIX, starting with column 1 and continuing for 23 characters.

```
% &PREFIX = &SUBSTR( &PREFIX, 1, 23)
```

Extract substring values from the values of &RANDOM and &STUFF, and assign the result to MY-LANGUAGE. Note that &QT defines the delimiter character as the apostrophe; the quotation marks delimiting the string values are not considered part of the string and are stripped from the output.

Input source:

```
% &QT = "'"
% &RANDOM = &SUBSTR( "MEANINGLESS", 1, 4)
% &STUFF = &SUBSTR( "EXAMPLE", 3, 5)
MY-LANGUAGE = &QT&RANDOM&STUFF&QT
```

Output source:

```
MY-LANGUAGE = 'MEANAMPLE'
```

Related Topics:	See...	For more information about...
	<i>% DECLARE</i>	Associative memory structures
	<i>% LOOKUP</i>	
	<i>% SET Statements</i>	Calling EPILOGUE macros
	<i>Sample Macros</i>	Examples: of functions

% IF/ELSE-IF/ELSE

Description: Evaluate a condition.

Syntax:

```
% IF condition1
statementblock
% ELSE-IF condition2
statementblock
% ELSE-IF conditionN
statementblock
% ELSE
statementblock
[% END]
```

Parameters: *condition*

Can be a:

Number, literal enclosed in apostrophes or quotation marks, or variable; followed by a space and a relational operator; followed by a space and a value. To specify multiple conditions, use the Boolean operators AND or OR with no parentheses.

Relational operator can be: =, NOT =, <, >, NOT <, NOT >, <=, >=.

Value can be a number, a literal enclosed in apostrophes or quotation marks, or a variable.

- Comments:**
- Indent each level of subordinate source code a consistent number of columns; we recommend four columns.
 - During processing, the Customization Facility moves statement blocks to the starting columns of their controlling % IF, % ELSE-IF, and % ELSE statements.
 - The limit of % ELSE-IFs for a single % IF is 199.

- This structure executes identically to the S-COBOL IF/ELSE-IF/ELSE structure. See the "IF/ELSE-IF/ELSE" topic in the APS Reference.

Examples: Test a variable for a numeric condition.

```
% IF &LEN = 80
% IF &APS-INDENT < 8
```

Test a variable for multiple conditions.

```
% IF &APS-INDENT < 8 OR &APS-INDENT > 11
```

Test a variable for a true condition, represented by the value 1.

```
% &EMPLOYEE-TYPE-A = 0
% &EMPLOYEE-TYPE-B = 1
% IF &EMPLOYEE-TYPE-B
  statementblock
```

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
```

Related Topics:	See...	For more information about...
	% <i>REPEAT</i>	Conditional processing
	% <i>UNTIL/WHILE</i>	
	% <i>IF Structure Example</i>	Examples: of conditional processing

% INCLUDE

Description: Open, read, and process a user-defined macro, copybook, or other file in an APS program.

Syntax: % INCLUDE *ddname*[(*membername*)] [*submember*]

Parameters: *ddname* Data set containing the file to include. Can be a partitioned data set, sequential file, or instream data set.

membername Member of *ddname*

submember Submember of *membername*

- Comments:**
- If you include a copybook that contains an indexed table, use % INCLUDE.
 - 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--such as SYM1--with an % INCLUDE to specify where to put the included file in the program.

- Examples:**
- Include a macro at the top of the program.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYM1   % INCLUDE USERMACS(MY-MACRO)
```

- Include a macro at the bottom of the program.

```
-KYWD- 12-*-----20---*-----30---*-----40---*-----50---*-----60
SYBT   % INCLUDE USERMACS(MY-MACRO)
```

- 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)
```

Related Topics:	See...	For more information about...
	<i>SY* Keywords</i>	Specifying in an APS program the location where the Customization Facility places source code

Limits, Customization Facility

APS enforces the following size and programming limitations.

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 of INCLUDE library	80
LRECL of extended INCLUDE library	140

Related Topics:	See...	For more information about...
	APS Reference: Limits	APS limits

% LOOKUP

Description: Reference a % DECLARE table.

Syntax:

```
% LOOKUP &declarepart oper searchval
          [FROM value [valueN ...]]
          statementblock
[% ELSE-IF condition
          statementblock
.
.
.
[% ELSE
          statementblock
[% END]
```

Parameters:	<i>&declarepart</i>	Name of variable assignment in % DECLARE table
	<i>condition</i>	Condition can be a number or variable, followed by a space and a relational operator, followed by a space and a value. To specify multiple conditions, use the Boolean operators AND or OR with no parentheses. Relational operator can be: =, NOT =, <, >, NOT <, NOT >, <=, >=.
	<i>oper</i>	Value can be a number, a literal enclosed in apostrophes or quotation marks, or a variable. Valid operator: =, NOT =, <, >, NOT <, NOT >, <=, >=.
	<i>searchval</i>	Customization Facility term.
	<i>value</i>	Starting value(s) of the low-order <i>&subscript</i> of the <i>&fieldname</i> associated with the <i>&declarepart</i> .

- Comments:**
- If FROM is not coded, the low-order *&subscript* in % DECLARE is the starting value and increments by 1 until the condition is satisfied or an undefined instance is found.
 - If the next-to-low-order DECLARE *&subscript* does not exist or is a string, then only the low-order *&subscript* enumerates by 1.

Examples: • Reference a % DECLARE table.

```

% DECLARE &IMS-PCB-( &PCBX )-SEG- ( &SEGX )
% &IMS-PCB-SEG-NAME                X30
% &IMS-PCB-SEG-IMSNAME              X8
% &IMS-PCB-SEG-PROCOPTS
% &IMS-PCB-SEG-PROCOPT-GET          N
% &IMS-PCB-SEG-PROCOPT-ISRT        N
% &IMS-PCB-SEG-PROCOPT-REPL        N
% &IMS-PCB-SEG-PROCOPT-DLET        N
% &IMS-PCB-SEG-LEN                  N6
.
.
% LOOKUP &IMS-PCB-SEG-NAME = &THE-SEG-NAME-YOU-WANT FROM 1 1
do this if found
% ELSE
do this if found

```

- Address *&declareparts* of other % DECLARE structures that use some or all of the same &subscripts without an additional % LOOKUP. Strip blanks from &ALIAS before the % LOOKUP by parsing it with the function &PARSE.

```

% DECLARE &VS-( &VSX-FILE )-KEY-( &VSX-KEY )-B
% &VSP-FILE-KEY-VOL                X8
% &VSP-FILE-KEY-SPACE              X20
% &VSP-FILE-KEY-CICZ               X20
% DECLARE &VS-( &VSX-FILE )-KEY-( &VSX-KEY )-ALIAS-( &VSX-
ALIAS )
% &VSP-FILE-KEY-ALIAS              X30
.
.
&ALIAS = &PARSE(&ALIAS)
.
.
% LOOKUP &VSP-FILE-KEY-ALIAS = &ALIAS FROM 1 1 1
% &VOL = &VSP-FILE-KEY-VOL

```

Related Topics:	See...	For more information about...
	% DECLARE	Defining % DECLARE tables

Macro Call

Description: Invoke a macro.

Syntax:

Format 1: `$macroname [(actualarg1[, actualarg2,
% ... actualarg3, ..., actualarg1000])]`

Format 2: `sourcecode <$macroname(actualarg1, ... actualarg1000)>
[... sourcecode]`

Parameters: *actualarg*

Actual argument list that corresponds to the formal argument list in the called macro's % DEFINE statement. Actual arguments pass values to their corresponding formal arguments.

Actual arguments are positional; if you omit any argument but the last one, code a comma in its place. Omitted arguments are called empty arguments.

Actualarg is a local variable and retains the values passed to it only from the time it is invoked until it ends.

Note: A variable defined outside the scope of a macro definition is global in scope.

An actual argument can be a:

- Number (maximum 7 digits), positive (default) or negative
- String enclosed by apostrophes or quotation marks
- Variable whose value is assigned in a variable assignment statement (see *Variable Assignment Statements*)
- Function (see *Functions, Customization Facility*)

- Comma, indicating an empty argument. You can assign a default value to an empty argument, as shown in the example below.
- Code the list according to the following rules:
- Separate arguments with a comma and optionally one or more spaces.
- Actual arguments are positional; if you omit any argument but the last one, code a comma in its place. Omitted arguments are called empty arguments.
- Enclose the argument list in parentheses.
- The maximum number of arguments in a list is 1000.

\$macroname

Macro to be invoked; the macro must be defined in the USERMACS PDS or data set in your Project and Group

- Comments:**
- Format 2 is an in-line macro call. You can embed an in-line macro call in any line of source code. When you do so, do the following:
 - Define a variable that has the same name as the in-line macro. Define this variable either before the in-line macro call, or at the top of the macro.
 - Assign a value to the variable. APS passes the value to the in-line macro during processing.
 - When you invoke an in-line macro, it appears immediately above the line that contains the in-line macro call, and in the same column as the first character of the line containing the in-line macro call. The rest of the macro retains relative positioning.
 - You can nest up to 139 macro calls in a macro call.

Examples: Pass the actual argument value 1 to the formal argument variable `&ARG1` in the macro, and the actual argument value 3 to `&ARG3`. In the macro, give `&ARG2` a default value of 0. Note that the extra comma between the actual arguments 1 and the 3 is a placeholder, indicating that the call does not pass an actual argument to `&ARG2`; it is an

undefined, or empty argument. Rather than leave the empty argument undefined, assign to it the default value 0. Also note that the call does not pass an actual argument to `&ARG4`. It, however, needs no placeholder because it is the last argument in the list; the macro assumes it is undefined.

- **Macro definition:**

```
% DEFINE $SAMPLE-MACRO( &ARG1, &ARG2, &ARG3,
... &ARG4)
    % IF NOT &DEFINED( &ARG2)
        % &ARG2 = "0"
    % END
.
.
.
% END
```

- **Macro call:**

```
$SAMPLE-MACRO( 1,, 3)
```

Note: Another way to assign a default value to an empty argument is to hard-code a variable assignment statement in the formal argument list as follows:

```
% DEFINE $SAMPLE-MACRO( &ARG1, &ARG2 = defaultvalue,
% ... &ARG3, &ARG4)
```

Pass a new value to `&NEXT-VALUE` each time that the in-line macro `$CNTR` is called.

```
% &CNTR = 0
% DEFINE $CNTR
    % &CNTR = &CNTR + 1
% END
.
.
% &VALUE-<$CNTR> = &NEXT-VALUE
.
.
% &VALUE-<$CNTR> = &NEXT-VALUE
```

Related Topics: See...

% *DEFINE*

For more information about...

Writing a macro % *DEFINE* statement

% REPEAT

Description: Establish a loop and test a condition.

Syntax:

Format 1:

```
% REPEAT
  statementblock
% UNTIL|WHILE condition
[ statementblock]
[% END]
```

Format 2:

```
% REPEAT VARYING|R-V &variable
% ... [FROM &variable|literal|arithmeticexpr]
% ... [BY &variable|literal|arithmeticexpr]
  statementblock
% ... UNTIL|WHILE condition
[ statementblock]
[% END]
```

Format 3:

```
% REPEAT VARYING|R-V &variable
% ... [FROM &variable|literal|arithmeticexpr]
% ... [BY &variable|literal|arithmeticexpr]
% ... THRU|TO &variable|literal|arithmeticexpr
[% ... OR THRU|TO &variable|literal|arithmeticexpr]
[% ... OR THRU|TO &variable|literal|arithmeticexpr]
[% ... OR THRU|TO &variable|literal|arithmeticexpr]
  statementblock
[% END]
```

Parameters:

condition

Condition can be a number or variable, followed by a space and a relational operator, followed by a space and a value. To specify multiple conditions, use the Boolean operators AND or OR with no parentheses.

Relational operator can be: =, NOT =, <, >, NOT <, NOT >, <=, >=.

Value can be a number, a literal enclosed in apostrophes or quotation marks, or a variable.

statementblock

Can contain any Customization Facility, COBOL or COBOL/2, or S-COBOL statements.

variable Customization Facility variable with valid COBOL name.

- Comments:**
- During processing, the Customization Facility moves % REPEAT statement blocks to the starting columns of their controlling % REPEAT statements.
 - The statement block subordinate to the % REPEAT, and any statement block subordinate to the % UNTIL or % WHILE, establish the loop. The loop executes until or while the condition is satisfied.
 - In formats 1 and 2, if the % UNTIL or % WHILE does not have a subordinate statement block, APS tests the condition and does the following:
 - If the % UNTIL condition is false or the % WHILE condition is true, the loop executes the % REPEAT statement block until the condition is true or while the condition is false, respectively.
 - If the % UNTIL condition is true or the % WHILE condition is false, the loop ends, and processing resumes at the next line that has the same or less indentation as the % UNTIL or % WHILE.
 - In formats 1 and 2, if the % UNTIL does have a subordinate statement block, APS tests the condition and does the following:
 - If the % UNTIL condition is false or the % WHILE condition is true, the loop executes the % UNTIL or % WHILE statement block and the % REPEAT statement block until the condition is true or while the condition is false, respectively.
 - If the % UNTIL condition is true or the % WHILE condition is false, the loop ends without executing the % UNTIL or % WHILE statement block, and processing resumes at the next line that has the same or less indentation as the % UNTIL or % WHILE.
 - In format 3, with the TO option, the loop executes to--but not including--its specified variable, literal, or arithmetic expression.
 - APS evaluates any evaluation bracket source in an % UNTIL and % WHILE statement block at each iteration of the loop; APS evaluates bracketed source in a % REPEAT and % REPEAT VARYING statement block just once.
 - If FROM is not coded, the default is the current value of the % REPEAT &*variable*.

- If BY is not coded, the default is BY 1.
- APS provides an internal loop counter, LOOP-LIMIT, which defaults to 500. To override the default, change the value of % SET LOOP-LIMIT in the APS CNTL file APSDBDC. You can override the default with a number as high as 999999. Be sure to reset the limit after an % INCLUDE statement. Note that some APS libraries set LOOP-LIMIT and reset it to 200 at the end.

Related Topics:	See...	For more information about...
	% UNTIL/WHILE	Testing % REPEAT loops
	Evaluation Brackets	Specifying the order in which the Customization Facility evaluates variable values
	Looping Example	Examples: of looping

% SET Statements

- Description:**
- Specify the program location where the Customization Facility places source code.
 - Override default processing.

- Syntax:**
- Program location statements:
 - % SET COMMUNICATION [SECTION]
 - % SET DATA [DIVISION]
 - % SET END-WORKING-STORAGE
 - % SET FILE-CONTROL
 - % SET FILE [SECTION]
 - % SET LINKAGE [SECTION]
 - % SET PROCEDURE
 - % SET SPECIAL-NAMES
 - % SET WORKING-STORAGE

- Processing override statements:

```
% SET AUXILIARY-OUTPUT
% SET NORMAL-OUTPUT

% SET BLANK
% SET NOBLANK

% SET CONVERT-LOWER-CASE
% SET PRESERVE-LOWER-CASE

% SET DELIMITERS-OPTIONAL
% SET DELIMITERS-REQUIRED

% SET EPILOGUE [$macroname]

% SET EVAL-BRACKETS 'leftright'
% SET EVAL-BRACKETS-AUX 'leftright'

% SET LEFT-MARGIN column
% SET RIGHT-MARGIN column

% SET ERROR message text
% SET FATAL message text
% SET INFO message text
% SET WARNING message text

% SET LOOP-LIMIT

SET TRACE ERROR [NOINFO]
SET NOTRACE

% SET WRITE-CONTROL[-LIMIT number]
% SET NOWRITE-CONTROL
```

Description:

NORMAL-OUTPUT |
AUXILIARY-OUTPUT

NORMAL-OUTPUT (default) directs all source code to the file with the external name POSTSOUT.

AUXILIARY-OUTPUT directs all source code to the file with the external name MACRO.

Note: You must define MACRO and its accompanying DD name in your JCL.

BLANK NOBLANK	<p>BLANK (default) prints all blank lines that appear in the input.</p> <p>NOBLANK suppresses blank lines from printing.</p>
COMMUNICATION [SECTION]	Moves all source code appearing between this statement and the next % SET statement from the Procedure Division to the end of the Communication Section.
PRESERVE LOWER-CASE CONVERT-LOWER-CASE	<p>PRESERVE-LOWER-CASE (default) allows developers to use both upper and lower case characters in Customization Facility structures.</p> <p>CONVERT-LOWER-CASE converts lower case characters in Customization Facility structures to upper case.</p>
DATA [DIVISION]	Moves all source code appearing between this statement and the next % SET statement from the Procedure Division to immediately after the Data Division header.
DELIMITERS-REQUIRED DELIMITERS-OPTIONAL	<p>DELIMITERS-REQUIRED (default) requires developers to delimit strings with apostrophes or quotation marks, depending on the value you set on the APS Precompiler Options window.</p> <p>DELIMITERS-OPTIONAL allows developers to omit delimiters around all strings except those that consist of all numbers or include special characters.</p>
END-WORKING-STORAGE	Marks a spot in Working-Storage to place source code that you move from the Procedure Division to Working-Storage using the % SET WORKING-STORAGE statement. Non-APS preprocessors can reserve the end of Working-Storage for their constructions.
ERROR FATAL WARNING INFO <i>messagetext</i>	Sends a message to the final message report of the compile. Enclose variables in <i>messagetext</i> with evaluation brackets.
EPILOGUE [<i>\$macroname</i>]	Calls <i>macroname</i> after all other Customization Facility processing finishes. Although you can specify only one macro call per EPILOGUE statement, you can write multiple statements. APS stores all EPILOGUE macro calls in an EPILOGUE queue, and invokes them in "last in, first called" order. The variable &APS-EPILOGUE contains the name of the macro call that is currently last in the EPILOGUE queue. Assuming that you code the EPILOGUE statement anywhere in the Procedure Division, APS places the invoked macros at the end of the Procedure Division.

	<p>To eliminate the current "last in" macro from the queue, follow the last % SET EPILOGUE statement with a % SET EPILOGUE statement, without specifying the macro name; APS gets the macro name value from &APS-EPILOGUE.</p> <p>Important: The results of calling APS macros in an EPILOGUE macro can be unpredictable. We recommend that you call only user-defined macros in an EPILOGUE macro.</p>
EVAL-BRACKETS ' <i>leftright</i> '	<p>Defines the characters to use as evaluation brackets. Default is < and >. We recommend using the default because that is what the APS macros use; we do not recommend using the parentheses characters as brackets.</p>
EVAL-BRACKETS-AUX ' <i>leftright</i> '	<p>Defines an auxiliary set of evaluation bracket characters to supplement the primary set defined by % SET EVAL-BRACKETS. Ensure that % SET EVAL-BRACKETS is also defined.</p>
FILE-CONTROL	<p>Moves all source code appearing between this statement and the next % SET statement from the Procedure Division to the end of the File-Control paragraph. APS generates the Input-Output Section and File-Control headers unless they are in the input source or if the macro is an EPILOGUE (see EPILOGUE above).</p>
FILE [SECTION]	<p>Moves all source code appearing between this statement and the next % SET statement from the Procedure Division to the end of the File Section. APS generates the File Section header unless it is in the input source or if the macro is an EPILOGUE (see EPILOGUE above).</p>
LEFT-MARGIN <i>columnnumber</i> RIGHT-MARGIN <i>columnnumber</i>	<p>Default: columns 1-72. Overrides the LANG option on the APS Precompiler Options window, which specifies the columns for the Customization Facility to process. Use these statements to specify the leftmost and rightmost columns to process, within the parameters of the value of LANG.</p>
LINKAGE [SECTION]	<p>Use in the Procedure Division to move all source code appearing between this statement and the next % SET statement to the end of the Linkage Section. APS generates the Linkage Section header unless it is in the input source or if the macro is invoked by % SET EPILOGUE (see EPILOGUE above).</p>

LOOP-LIMIT	Overrides the internal loop counter default (500) in the APS CNTL file APSDBDC. Override with a number as high as 999999. Because some APS libraries reset LOOP-LIMIT to 200, you might need to reset the limit after an % INCLUDE statement.
PROCEDURE	Mark the end of the preceding relocation statement and return to the location of the statement (not the beginning or end of the Procedure Division).
SPECIAL-NAMES	Use in the Procedure Division to move all source code appearing between this statement and the next % SET statement to the end of the Special-Names paragraph. APS generates the Special-Names header unless it is in the input source or if the macro is an EPILOGUE (see EPILOGUE above).
TRACE ERROR [NOINFO]	TRACE ERROR traces all source code appearing between this statement and NOTRACE and reports on all errors of all severity levels--F(atal), E(rror), W(arning), and I(nformation). TRACE ERROR displays error messages and other information including: the line of source that caused the error; active % INCLUDE statements; active macros; and the number of loops completed at the time of error (if applicable) TRACE ERROR NOINFO traces all levels of errors except I(nformation).
WORKING-STORAGE	Use in the Procedure Division to move all source code appearing between this statement and the next % SET statement to the end of the Working-Storage Section or immediately in front of the % SET END-WORKING-STORAGE statement, if coded.
WRITE-CONTROL [-LIMIT <i>number</i>] and NOWRITE-CONTROL	WRITE-CONTROL prints in the output all Customization Facility source code appearing between this statement and % SET NOWRITE-CONTROL; default is 5000 lines.

Examples: Store three macro calls in the EPILOGUE queue, but invoke just \$MACRO-1.

```
% SET EPILOGUE $MACRO-1
% SET EPILOGUE $MACRO-2
% SET EPILOGUE $MACRO-3
    %* Current value of &APS-EPILOGUE is $MACRO-3.
.
.
```

```

.
% SET EPILOGUE
  %* $MACRO-3 (the current value of &APS-EPILOGUE) is
  %* eliminated from the EPILOGUE queue;
  %* the new current value of &APS-EPILOGUE is $MACRO-2.
.
.
.
% SET EPILOGUE
  %* $MACRO-2 (the current value of &APS-EPILOGUE) is
  %* eliminated from the EPILOGUE queue;
  %* the new current value of &APS-EPILOGUE is $MACRO-1.
  %* $MACRO-1 (the current value of &APS-EPILOGUE) is
  %* invoked.

```

Related Topics:	See...	For more information about...
	<i>SY* Keywords</i>	Specifying in an APS program the location where the Customization Facility places source code
	<i>Evaluation Brackets</i>	Specifying the order in which the Customization Facility evaluates variable values
	<i>Program Location Statements Example</i>	Examples: of program location statements

SY* Keywords

Description: Specify in an APS program the location where the Customization Facility places source code.

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

SYBT *macrocode*

SYEN *macrocode*

SYDD *macrocode*

SYFD *macrocode*

SYIO *macrocode*

SYLT *macrocode*

SYLK *macrocode*

SYM1 *macrocode*

SYM2 *macrocode*

SYRP *macrocode*

SYWS *macrocode*

Locations in Generated Code

SYM1 At the beginning of the program, before macro libraries that you include at the beginning of the program

SYM2 After macro 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 macro libraries that you include at the beginning of the Input-Output Section

Data Division

SYDD At the beginning of the Data Division

SYFD In the File Section, after macro libraries that you include at the beginning of the File Section

SYWS In the Working-Storage Section, after macro libraries and data structures that you include in Working-Storage

SYLT In the Linkage Section, after macro 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 macro 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 macrocode to start in column 8.

Examples:

- Include a macro at the top of the program.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
      SYM1  % INCLUDE USERMACS(MY-MACRO)
```

- Include a macro at the bottom of the program.

```
-KYWD- 12-*----20---*----30---*----40---*----50---*----60
      SYBT  % INCLUDE USERMACS(MY-MACRO)
```

- 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)
```

Related Topics:

See...	For more information about...
% <i>SET</i> Statements	Specifying the program location where the Customization Facility places source code
% <i>INCLUDE</i>	Including macros or copybooks in a macro
APS Reference: "Keywords"	Other APS keywords

% UNTIL/WHILE

Description: Establish a loop and test a condition to execute a subordinate statement block either until or while a single or compound condition is satisfied.

Syntax: % UNTIL|WHILE *condition*
 statementblock
 [% END]

Parameter: *condition* Valid values:

- Constant numeric term
- Relational operator can be: =, NOT =, <, >, NOT <, NOT >, <=, >=.
- Variable that has a numeric true/false value

AND or OR may connect multiple logical terms. Do not use parentheses.

- Comments:**
- During processing, the Customization Facility moves % UNTIL and % WHILE statement blocks to the starting columns of their controlling % UNTIL and % WHILE statements.
 - The % UNTIL statement block executes until the condition is true. Conversely, the % WHILE statement block executes while the condition is false.
 - When the % UNTIL condition is true or the % WHILE condition is false, the loop ends without executing the % UNTIL or % WHILE statement block, and processing resumes at the next line that has the same or less indentation as the % UNTIL or % WHILE.
 - APS provides an internal loop counter, LOOP-LIMIT, which defaults to 500. To override the default, change the value of % SET LOOP-LIMIT in the APS CNTL file APSDBDC. You can override the default with a number as high as 999999. Be sure to reset the limit after an % INCLUDE statement. Note that some APS libraries set LOOP-LIMIT and reset it to 200 at the end.
 - APS evaluates any evaluation bracket source in an % UNTIL and % WHILE statement block at each iteration of the loop; APS evaluates bracketed source in a % REPEAT and % REPEAT VARYING statement block just once.
 - Use % UNTIL or % WHILE with a % REPEAT loop to add a conditional test to the % REPEAT loop.

Related Topics:	See...	For more information about...
	% REPEAT	Using % UNTIL or % WHILE with a % REPEAT loop
	Looping Example	Examples: of looping

Variable Assignment Statements

Description: Assign a value to a Customization Facility variable.

Syntax: `% &variable = stringterm|numericterm`

Parameters: *numericterm* Numeric term can be one or a combination of:

- A number (maximum 7 digits), positive (default) or negative
- A number that is a true/false representation of a variable, where 0 = false and nonzero = true
- An arithmetic expression with arithmetic operators listed in order of processing: * (multiplication), / (division), + (addition), - (subtraction)
- A Customization Facility number function

variable Valid COBOL name

stringterm String term can be:

- Text enclosed by apostrophes or quotation marks
- A *&variable* with a string value
- A Customization Facility string function

Examples: • Assign the literal value SPACE to the variable &TEST-CHR.

```
% &TEST-CHR = 'SPACE'
```

• Assign the APS substringing function &SUBSTR to the variable &A.

```
% &A = &SUBSTR( &B, 3, 5)
```

• Assign the numeric value 121 to the variable &REC-LEN.

```
% &REC-LEN = 121
```

• Assign the variable &B, whose value is a number, to the variable &A.

```
% &B = 100
```

```
% &A = &B
```

- Assign the representation of true to the variable &EMPLOYEE-TYPE-B and test whether that variable is true.

```
% &EMPLOYEE-TYPE-A = 0
% &EMPLOYEE-TYPE-B = 1
% IF &EMPLOYEE-TYPE-B
```

- Assign the value of &B + &C to the variable &A. Assume the values of &B and &C are numeric.

```
% &A = &B + &C
```

- Divide &LINELENGTH by 2, divide &NN by 2, and subtract the result of &NN / 2 from the result of &LINELENGTH / 2. Assign the final result as the value of &COL.

```
% &COL = &LINELENGTH / 2 - &NN / 2
```

- Concatenate two strings, &B and &C, and assign the result as the value of &A. Rather than delimiting &B and &C with apostrophes, delimit them with variables that have the value of an apostrophe so that the Customization Facility processor does not process '&B&C' as a string.

```
% &SQ = " ' "
% &B = ' CON '
% &C = ' CAT '
% &A = <% &SQ&B&C&SQ>
```

Related Topics: See...

*Functions,
Customization Facility*

For more information about...

Using Customization Facility functions

3 Sample Macros

This chapter contains sample macros you can use to customize APS.

% IF Structure Example

This sample macro uses the % IF structure to build a program that tests records for any specific character and replaces some occurrences of it with any other specific character, depending on the character's mode (for example, whether the character appears as a leading or trailing character, or anywhere). The variables that control what you search for and replace with (and the length of the record read) appear at the top of the program. Thus, you can use this macro many different times for many different purposes simply by changing the values of a few variables.

Input source:

```

% &MODE = 'TRAILING'                201.
% &REC-LEN = 121                    202.
% &TEST-CHR = 'SPACE'              203.
% &REPLACE-CHR = 'ZERO'            204.
IDENTIFICATION DIVISION.           205.
PROGRAM-ID.                          EXAMPLE2. 206.
*SPECIAL CONSIDERATIONS.            207.
* SAMPLE PROGRAM TO READ RECS WHOSE LENGTH IS DETERMINED AT 208.
* AT COMPILE-TIME & REPLACE SOME OCCURRENCES OF TEST-CHR 209.
* BY THE REPLACE-CHR (BOTH TO BE SET AT COMPILE-TIME).    210.
* REPLACEMENT MODE USED (LEADING, TRAILING, OR ALL)       211.
* IS SET AT COMPILE-TIME.                                212.
                                        213.
ENVIRONMENT DIVISION.                214.
INPUT-OUTPUT SECTION.                215.
FILE-CONTROL.                         216.
    SELECT INPUT-FILE                 ASSIGN-UT-S-INPUT. 217.
    SELECT OUTPUT-FILE                ASSIGN-UT-S-OUTPUT. 218.
                                        219.

```

```

DATA DIVISION.                                220.
FILE SECTION.                                  221.
                                                222.
FD INPUT-FILE                                  223.
   BLOCK CONTAINS 0 RECORDS                    224.
   LABEL RECORDS ARE STANDARD.                 225.
01 INPUT-RECORD                                226.
   PIC X(&REC-LEN).                             227.
                                                228.
FD OUTPUT-FILE                                  229.
   BLOCK CONTAINS 0 RECORDS                    229.
   LABEL RECORDS ARE STANDARD.                 230.
01 OUTPUT-RECORD                               231.
   PIC X(&REC-LEN).                             232.
WORKING-STORAGE SECTION.                       233.
                                                234.
01 WS-RECORD.                                  235.
   02 WS-CHR                                    236.
   PIC X OCCURS &REC-LEN.                      237.
01 II                                           238.
   PIC S9(3) COMP SYNC.                        239.
PROCEDURE DIVISION.                            240.
                                                241.
OPEN INPUT INPUT-FILE                          242.
OPEN OUTPUT OUTPUT-FILE                       243.
                                                244.
REPEAT                                          245.
   READ INPUT-FILE INTO WS-RECORD              246.
UNTIL AT END ON INPUT-FILE                     247.
   % IF &MODE = 'LEADING'                      248.
   REPEAT VARYING II FROM 1 BY 1               249.
   UNTIL II > &REC-LEN                         250.
   ... OR WS-CHR (II) NOT = &TEST-CHR          251.
   WS-CHR (II) = &REPLACE-CHAR                252.
   % ELSE-IF &MODE = 'TRAILING'                253.
   REPEAT VARYING II FROM &REC-LEN BY -1       254.
   UNTIL II < 1                                255.
   ... OR WS-CHR (II) NOT = &TEST-CHR          256.
   WS-CHR (II) = &REPLACE-CHAR                257.
   % ELSE-IF &MODE = 'ALL'                     258.
   REPEAT VARYING II FROM 1 BY 1               259.
   UNTIL II > &REC-LEN                         260.
   IF WS-CHR (II) = &TEST-CHR                  261.
   WS-CHR (II) = &REPLACE-CHAR                262.
   % ELSE                                       263.
   DISPLAY "PARAMETER ERROR IMPROPER MODE:" &MODE 264.
                                                265.
   WRITE OUTPUT-RECORD FROM WS-RECORD         266.
                                                267.
CLOSE INPUT-FILE OUTPUT-FILE                  268.

```

Output source:

```

020500 IDENTIFICATION DIVISION.                205.
020600 PROGRAM-ID.                            EXAMPLE2.    206.
020700*SPECIAL CONSIDERATIONS.                207.
020800* PROGRAM TO READ RECS WHOSE LENGTH IS DETERMINED AT 208.
020900* COMPILE-TIME & REPLACE SOME OCCURRENCES OF TEST-CHR 209.
021000* BY THE REPLACE-CHR (BOTH TO BE SET AT COMPILE-TIME). 210.
021100* REPLACEMENT MODE USED (LEADING, TRAILING OR ALL)    211.
021200* IS SET AT COMPILE-TIME.                212.
021300                                         213.
021400 ENVIRONMENT DIVISION.                  214.
021500 INPUT-OUTPUT SECTION.                  215.
021600 FILE-CONTROL.                          216.
021700     SELECT INPUT-FILE                   ASSIGN UT-S-INPUT. 217.
021800     SELECT OUTPUT-FILE                  ASSIGN UT-S-OUTPUT. 218.
021900                                         219.
022000 DATA DIVISION.                        220.
022100 FILE SECTION.                          221.
022200                                         222.
022300 FD  INPUT-FILE                          223.
022400     BLOCK CONTAINS 0 RECORDS            224.
022500     LABEL RECORDS ARE STANDARD.         225.
022600 01  INPUT-RECORD                        PIC X(121).    226.
022700                                         227.
022800 FD  OUTPUT-FILE                         228.
022900     BLOCK CONTAINS 0 RECORDS            229.
023000     LABEL RECORDS ARE STANDARD.         230.
023100 01  OUTPUT-RECORD                      PIC X(121).    231.
023200                                         232.
023300 WORKING-STORAGE SECTION.               233.
023400                                         234.
023500 01  WS-RECORD.                          235.
023600 02  WS-CHR                              PIC X OCCURS 121. 236.
023700                                         237.
023800 01  II                                PIC S9(3) COMP SYNC. 238.
023900                                         239.
024000 PROCEDURE DIVISION.                    240.
024100                                         241.
024200     OPEN INPUT INPUT-FILE                242.
024300     OPEN OUTPUT OUTPUT-FILE            243.
024400                                         244.
024500     REPEAT                                245.
024600         READ INPUT-FILE INTO WS-RECORD    246.
024700     UNTIL AT END ON INPUT-FILE            247.
025400         REPEAT VARYING II FROM 121 BY -1 254.
025500         UNTIL II < 1                      255.
025600         ... OR WS-CHR (II) NOT = SPACE    256.

```

```

025700          WS-CHR (II) = ZERO                257.
026600          WRITE OUTPUT-RECORD FROM WS-RECORD 266.
026700                                                267.
026800          CLOSE INPUT-FILE OUTPUT-FILE      268.

```

Discussion

- Lines 201-204 of Input. Variables are assigned values. These variables are used in the Procedure Division to replace any trailing space with a zero.
- Lines 245-247 and 266-268.
 - Input: The S-COBOL REPEAT verb establishes a loop that can be tested at the middle or end. The statement blocks subordinate to REPEAT and UNTIL form the loop and are executed repeatedly under control of the condition specified by the UNTIL statement (for example, until the end of file). At end of file, processing continues at line 268. After line 268, S-COBOL generates a STOP RUN.
 - Output: These lines are copied into the output source because they contain no Customization Facility syntax.
- Lines 253-257.
 - Input: This statement block is executed because &MODE has been assigned the value 'TRAILING' at the top of the program. REPEAT VARYING ... UNTIL ... is an extension of the S-COBOL REPEAT verb. It establishes a test and causes its subordinate statement block to be executed repeatedly until the test is true.
 - Output: Line 253 does not appear in the output because it is a Customization Facility statement. The other lines do appear. The variables &REC-LEN, &TEST-CHR, and &REPLACE-CHR are resolved with their values 121, SPACE, and ZERO. The loop starts at the 121st character, is decremented by 1, and executes line 257 until a trailing character appears that is not a space or until all 121 characters have been tested.
- Lines 263-265.
 - Input: If &MODE is neither LEADING, TRAILING, or ALL, control falls to this % ELSE statement and displays an error message.

- Output: Note that input line 265 is blank. It does not appear in the output because it is considered part of the statement block subordinate to % ELSE. Remember: a statement block ends with the first non-blank character at the same or less indentation than the controlling conditional statement. To get the blank line to appear in the output, end the conditional structure by inserting a line beneath line 264 and coding % END at the same indentation as % ELSE; then leave the next line blank.

Looping Example

This program calls two user macros that use loops to create a data structure. A loop that incorporates built-in functions returns a value to text.

User macros:

```

%* HERE ARE TWO MACROS: $ITEM-MAKER AND $TABLE-MAKER           601.
                                                                602.
% DEFINE $ITEM-MAKER( &MM, &DATANAME, &TAIL)                   603.
  % &II = 1                                                       604.
  % WHILE &II <= &MM                                             605.
    02 &DATANAME+-&II          &40+PIC &TAIL.                  606.
    % &II = &II + 1                                             607.
% END                                                            608.
                                                                609.
% DEFINE $TABLE-MAKER( &MM, &NN, &TNAME, &INAME, &TAIL)      610.
  % &II = 1                                                       611.
  % WHILE &II <= &MM                                             612.
    02 &TNAME+-&II.                                             613.
    % &JJ = 1                                                   614.
    % UNTIL &JJ > &NN                                           615.
      % BEGIN                                                  616.
        03 &INAME+-&II+-&JJ &40+PIC &TAIL.                    617.
        % &JJ = &JJ + 1                                         618.
      % &II = &II + 1                                           619.
% END                                                            620.
                                                                621.

```

Program input:

```

% SET BLANK 622.
IDENTIFICATION DIVISION. 623.
PROGRAM-ID. EXAMPLE6. 624.
*SPECIAL CONSIDERATIONS. 625.
* DEMONSTRATE LOOPING. 626.
627.
----- 628.
----- 629.
630.
WORKING-STORAGE SECTION. 631.
632.
01 TEST-RECORD. 633.
 $ITEM-MAKER( 12, 'TEST-ITEM', 'S9(9) COMP SYNC VALUE -1' ) 634.
 $TABLE-MAKER( 3, 4, 'ENTRY', 'ITEM', 'X(4) VALUE SPACES' ) 635.
% &STR = 'MISSISSIPPI' 637.
% &SUB = 'IS' 638.
% &II = 0 639.
% REPEAT 640.
  % &JJ = &INDEX( &STR, &SUB ) 641.
% UNTIL &JJ = 0 642.
  % &II = &II + 1 643.
  % &JJ = &JJ + &LENGTH( &SUB ) 644.
  % &STR = &SUBSTR( &STR, &JJ ) 645.
THE NUMBER OF 'IS' IN 'MISSISSIPPI' IS &II. 646.
647.

```

Program output:

```

062300 IDENTIFICATION DIVISION. 623.
062400 PROGRAM-ID. EXAMPLE6. 624.
062500*SPECIAL CONSIDERATIONS. 625.
062600* DEMONSTRATE LOOPING. 626.
062700 627.
062800 ----- 628.
062900 ----- 629.
063000 630.
063100 WORKING-STORAGE SECTION. 631.
063200 632.
063300 01 TEST-RECORD. 633.
063400 02 TEST-ITEM-1 PIC S9(9) COMP SYNC VALUE -1. 634.
063402 02 TEST-ITEM-2 PIC S9(9) COMP SYNC VALUE -1. 623.
063404 02 TEST-ITEM-3 PIC S9(9) COMP SYNC VALUE -1. 634.
063406 02 TEST-ITEM-4 PIC S9(9) COMP SYNC VALUE -1. 634.
063408 02 TEST-ITEM-5 PIC S9(9) COMP SYNC VALUE -1. 634.

```

```

063410 02 TEST-ITEM-6          PIC S9(9) COMP SYNC VALUE -1. 634.
063412 02 TEST-ITEM-7          PIC S9(9) COMP SYNC VALUE -1. 634.
063414 02 TEST-ITEM-8          PIC S9(9) COMP SYNC VALUE -1. 634.
063416 02 TEST-ITEM-9          PIC S9(9) COMP SYNC VALUE -1. 634.
063418 02 TEST-ITEM-10         PIC S9(9) COMP SYNC VALUE -1. 634.
063420 02 TEST-ITEM-11         PIC S9(9) COMP SYNC VALUE -1. 634.
063422 02 TEST-ITEM-12         PIC S9(9) COMP SYNC VALUE -1. 634.
063500 02 ENTRY-1.             635.
063502 03 ITEM-1-1             PIC X(4) VALUE SPACES.        635.
063504 03 ITEM-1-2             PIC X(4) VALUE SPACES.        635.
063506 03 ITEM-1-3             PIC X(4) VALUE SPACES.        635.
063508 03 ITEM-1-4             PIC X(4) VALUE SPACES.        635.
063510 02 ENTRY-2.             635.
063512 03 ITEM-2-1             PIC X(4) VALUE SPACES.        635.
063514 03 ITEM-2-2             PIC X(4) VALUE SPACES.        635.
063516 03 ITEM-2-3             PIC X(4) VALUE SPACES.        635.
063518 03 ITEM-2-4             PIC X(4) VALUE SPACES.        635.
063520 02 ENTRY-3.             635.
063522 03 ITEM-3-1             PIC X(4) VALUE SPACES.        635.
063524 03 ITEM-3-2             PIC X(4) VALUE SPACES.        635.
063526 03 ITEM-3-3             PIC X(4) VALUE SPACES.        635.
063528 03 ITEM-3-4             PIC X(4) VALUE SPACES.        635.
063600                                     636.
064600 THE NUMBER OF 'IS' IN 'MISSISSIPPI' IS 2.        646.
064700                                     647.

```

Discussion

- Macro lines 603-608. The macro \$ITEM-MAKER is defined with three formal arguments whose values are supplied by the macro call in the program on line 634. % WHILE starts a loop that is executed repeatedly while the value of &I is <= to the value of &MM. The loop generates the 02 TEST-ITEM data elements (see output line 634). Note the two uses of the plus symbol in line 606. Used in &DATANAME+-&I, the plus concatenates the two variables, resulting in TEST-ITEM-1 for the first data item. On the next line, &I is incremented by one during each loop, resulting in TEST-ITEM-2 for the second data item, etc. The second use for the plus symbol is to place the value of PIC &TAIL in column 40 of the output for each TEST-ITEM.
- Macro lines 610-620. The macro \$TABLE-MAKER is defined with five formal arguments whose values are supplied by the macro call in the program on line 635. % WHILE starts an outer loop that repeatedly executes lines 613-619 while the value of &I is <= the

value of &MM. The outer loop generates the 02 data elements ENTRY-1, ENTRY-2, and ENTRY-3. & UNTIL starts a nested, inner loop. Its subordinate statement block (lines 616-618) is executed repeatedly until the value of &JJ is > the value of &NN. The inner loop generates the 03 data elements for the outer loop 02s. Note that the % BEGIN statement means that the 03 data items will appear in the output shifted left four spaces in the inner loop so that they are indented two spaces from the 02 data items. Without % BEGIN, they would be at the same indentation as the 02s. (The whole outer and inner loop structure shifts as a unit, thus all lines retain their relative positioning.) Line 618 increments the inner loop variable &JJ. Line 619 increments the outer loop variable &I; note that to be part of the outer loop, it must be indented from the outer loop but not from the inner loop. The loops generate the multiple lines output from both loops appears in the multiple lines 634-635.

- Line 622, Input. % SET BLANK means that all blank lines that follow appear in the output
- Lines 637-646, Input. These lines illustrate a % REPEAT and % UNTIL structure that uses the built-in functions &INDEX, &LENGTH, and &SUBSTR to find the number of occurrences of the substring IS in the literal string MISSISSIPPI. Lines 641-645 are executed repeatedly until &JJ = 0.

Customization Facility Functions Example

User macro:

```

%* HERE IS THE MACRO $DAYDEF                                501.
                                                            502.
% DEFINE $DAYDEF( &DAY)                                     506.
  % &PREFIX = &SUBSTR( &DAY, 1, 24)                         507.
  % &CHR = &SUBSTR( &PREFIX, 24)                            508.
  % IF &CHR = '-'                                           509.
    % &PREFIX = &SUBSTR( &PREFIX, 1, 23)                    510.
  % &SUFFIX = '-ENTRY'                                       511.
  % &DAYCTR = &DAYCTR + 1                                    512.
02  &PREFIX&SUFFIX.                                         513.
    03  FILLER    &40+PIC S9(2) COMP SYNC VALUE +&LENGTH( &DAY).514.

```

```

% IF &LENGTH( &DAY) <= 16                                515.
  03 FILLER &40+PIC X(30) VALUE &QT&DAY&QT.              516.
% ELSE                                                      517.
  03 FILLER &40+PIC X(30) VALUE                          518.
      &40+&QT&DAY&QT.                                    519.
% END                                                       520.
                                                         521.

```

Program input:

```

% &DAYCTR = 0                                             522.
% &QT = " "                                              523.
                                                         524.
% SET BLANK                                              525.
IDENTIFICATION DIVISION.                                526.
PROGRAM-ID.                EXAMPLE&INDEX( 'ABCDEFG', 'EF'). 527.
DATE-COMPILED.             &COMPILETIME.                528.
*SPECIAL CONSIDERATIONS.  529.
*   A PIECE OF A SAMPLE PROGRAM TO DEMONSTRATE SOME     530.
*   OF THE MACRO FACILITY BUILT-IN FUNCTIONS.          531.
                                                         532.
   --                                                    533.
   --                                                    534.
                                                         535.
WORKING-STORAGE SECTION.                                536.
                                                         537.
01 DAY-TABLE.                                           538.
   $DAYDEF( 'SUNDAY' )                                   539.
   $DAYDEF( 'MONDAY' )                                   540.
   $DAYDEF( 'TUESDAY' )                                  541.
   $DAYDEF( 'WEDNESDAY' )                               542.
   $DAYDEF( 'THURSDAY' )                                543.
   $DAYDEF( 'FRIDAY' )                                  544.
   $DAYDEF( 'A-LARGE-SYMBOL-TO-CHECK-DAYDEF' )         545.
01 DAY-TABLE-REDEF REDEFINES DAY-TABLE.                546.
  02 DAY-ENTRY OCCURS &DAYCTR.                          547.
    03 DAY-LEN PIC S9(2) COMP SYNC.                     548.
    03 DAY-BOL PIC X(30).                               549.
                                                         550.
PROCEDURE DIVISION.                                    551.
                                                         552.
   --                                                    553.
   --                                                    554.

```

Program output:

```

052300 IDENTIFICATION DIVISION.                                526.
052400 PROGRAM-ID.                EXAMPLE5.                    527.
052500 DATE-COMPILED.              22 OCT 79 14.24.00.        528.
052600*SPECIAL CONSIDERATIONS.                                          529.
052700*   A PIECE OF A SAMPLE PROGRAM TO DEMONSTRATE SOME          530.
052800*   OF THE MACRO FACILITY BUILT-IN FUNCTIONS.                531.
052900                                          532.
053000   --                                          533.
053100   --                                          534.
053200                                          535.
053300 WORKING-STORAGE SECTION.                                         536.
053400                                          537.
053500 01 DAY-TABLE.                                                  538.
053600 02 SUNDAY-ENTRY.                                              539.
053602 03 FILLER                PIC S9(2) COMP SYNC VALUE +6  539.
053604 03 FILLER                PIC X(30) VALUE 'SUNDAY'.      539.
053700 02 MONDAY-ENTRY.                                              540.
053702 03 FILLER                PIC S9(2) COMP SYNC VALUE +6.  540.
053704 03 FILLER                PIC X(30) VALUE 'MONDAY'.      540.
053800 02 TUESDAY-ENTRY.                                             541.
053802 03 FILLER                PIC S9(2) COMP SYNC VALUE +7.  541.
053804 03 FILLER                PIC X(30) VALUE 'TUESDAY'.     541.
053900 02 WEDNESDAY-ENTRY.                                           542.
053902 03 FILLER                PIC S9(2) COMP SYNC VALUE +9.  542.
053904 03 FILLER                PIC X(30) VALUE 'WEDNESDAY'.   542.
054000 02 THURSDAY-ENTRY.                                            543.
054002 03 FILLER                PIC S9(2) COMP SYNC VALUE +8.  543.
054004 03 FILLER                PIC X(30) VALUE 'THURSDAY'.    543.
054100 02 FRIDAY-ENTRY.                                              544.
054102 03 FILLER                PIC S9(2) COMP SYNC VALUE +6.  544.
054104 03 FILLER                PIC X(30) VALUE 'FRIDAY'.      544.
054200 02 A-LARGE-SYMBOL-TO-CHECK-ENTRY.                             545.
054202 03 FILLER                PIC S9(2) COMP SYNC VALUE +30. 545.
054204 03 FILLER                PIC X(30) VALUE
054206                                'A-LARGE-SYMBOL-TO-CHECK-DAYDEF'. 545.
054300 01 DAY-TABLE-REDEF REDEFINES DAY-TABLE.                    546.
054400 02 DAY-ENTRY OCCURS 7.                                        547.
054500 03 DAY-LEN                PIC S9(2) COMP SYNC.          548.
054600 03 DAY-BOL                PIC X(30).                     549.
054700                                          550.
054800 PROCEDURE DIVISION.                                           551.
054900                                          552.
055000   --                                          553.
055100   --                                          554.

```

Discussion

The \$DAYDEF Macro

\$DAYDEF uses the built-in functions &SUBSTR and &LENGTH to build a data structure for a table. \$DAYDEF builds for each table entry an 02-level data name and provides the values of the VALUE clauses of the two 03-level entries that follow it. The 02-level data name is constructed as &PREFIX&SUFFIX.

The value of &SUFFIX for each macro call is the literal string -ENTRY.

The value of &PREFIX is supplied by the formal argument &DAY which receives its value from a macro call to \$DAYDEF. For example, the first macro call in the program gives the value SUNDAY to &DAY. The built-in function &SUBSTR checks this value and dictates that up to 24 characters will be accepted (values longer than 24 characters are truncated). On the next line, &SUBSTR checks to see whether the 24th character (if there is one) is a hyphen. If it is a hyphen, only 23 characters are used in order to prevent a double hyphen which would be undesirable (for example, you don't want a hyphen from the name followed by the hyphen of the suffix -ENTRY). SUNDAY is only 6 characters long, so the entire value is accepted as the value for &PREFIX. Thus the value of &PREFIX&SUFFIX for the first call is SUNDAY-ENTRY.

Now \$DAYDEF provides the values of the VALUE clauses of the two 03-level data elements for SUNDAY-ENTRY. In the first 03 element, VALUE +&LENGTH(&DAY) becomes VALUE +6 because the value of &DAY (which is SUNDAY) is 6 characters long. The plus symbol is recognized as the COBOL plus symbol because there is a space before it. (In contrast, the plus symbol in &40+PIC is recognized as a Customization Facility column indicator, which in this case means "Put the PIC clause in column 41.")

Line 515 checks to see whether the length of the values of &DAY can fit all on one line (for example, whether it is 16 characters maximum). If it can fit, its format is the one on line 516 (remember: the line will shift under the % IF statement during processing). If it cannot fit, its format is the two-line format on lines 518-519.

The Program:

- Lines 527-528, Input. &INDEX returns a number which identifies the character position of the first occurrence of EF within ABCDEFG. The number is 5 -- thus EXAMPLE&INDEX('ABCDEFGF', 'EF') becomes EXAMPLE5. This is simply an illustrative example, not a particularly useful one. &COMPILETIME, however, returns the current date and time each time the program is compiled.
- Lines 539-545, Input. These are seven macro calls to \$DAYDEF, each providing a value for the &DAY argument in \$DAYDEF.
- Line 547. The OCCURS clause value &DAYCTR is initialized to 0 at the top of the program and is used in \$DAYDEF.

Program Location Statements Example

This example illustrates the use of program location statements and in-line macros. In addition to showing macros, program input and the resultant Customization Facility output, this example goes one step further and shows the program after APS Precompiler processing.

User macros:

```

1.          % DEFINE $IN-LINE( &ARG)
2.              COUNTER = &ARG
3.          % &IN-LINE = &ARG
4.          % END
5.          % DEFINE $MY-EPILOGUE
6.              % * MARK COLUMN 7 MARGIN
7.              % SET LINKAGE
8.              01 MY-EPILOGUE-CHR          &40+PIC X.
9.              % SET PROCEDURE
10.                 COUNTER = 100
11.                 % SET EPILOGUE $SECOND-EPILOGUE
12.          % DEFINE $SECOND-EPILOGUE
13.              % BEGIN
14.                 COUNTER = 101
15.          % SET EPILOGUE $MY-EPILOGUE

```

Program input:

```

16.          % SET EVAL-BRACKETS "<>"
17.          IDENTIFICATION DIVISION.
18.          PROGRAM-ID.
19.                                     EXAMPLE9.
20.          ENVIRONMENT DIVISION.
21.          INPUT-OUTPUT SECTION.
22.          FILE-CONTROL.
23.              SELECT YOUR-FILE          ASSIGN UT-S-YOURFILE.
24.          DATA DIVISION.
25.          FILE SECTION.
26.          FD  YOUR-FILE.
27.          01  YOUR-RECORD                PIC X(80) .
28.          WORKING-STORAGE SECTION.
29.          01  COUNTER                    PIC S9(4) COMP.
30.          01  WS-1                       PIC X.
31.          % SET END-WORKING-STORAGE
32.          *  PLACE FOR STUFF THAT MUST BE AT END OF WS FOR
33.          *  CERTAIN NON-APS PRE-PROCESSORS.
34.          PROCEDURE DIVISION.
35.              COUNTER = 1
36.              $IN-LINE( 2)
37.              TEST-NO-DELIMITERS = <$IN-LINE( 3)>
38.              ADD <$IN-LINE( 4)> <$IN-LINE( 5)> TO COUNTER
39.          % SET LINKAGE
40.          01  LINKAGE-STUFF              PIC X.
41.          % SET PROCEDURE
42.          % SET DELIMITERS-OPTIONAL
43.              $IN-LINE( TEST-NO-DELIMITERS)
44.              $IN-LINE( 6)
45.          % SET DATA
46.          *  PLACING DATA IN DATA DIVISION
47.          % SET PROCEDURE
48.          % SET FILE
49.          FD  MY-FILE.
50.          01  MY-RECORD                  PIC X(80) .
51.          % SET PROCEDURE
52.          % SET FILE-CONTROL
53.              SELECT MY-FILE            ASSIGN UT-S-MYFILE.
54.          % SET PROCEDURE
55.              OPEN INPUT MY-FILE YOUR-FILE
56.              CLOSE MY-FILE YOUR-FILE
57.          % SET WORKING-STORAGE
58.          01  TEST-NO-DELIMITERS        PIC 9(&IN-LINE) .
59.          % SET PROCEDURE
61.          % SET AUXILIARY-OUTPUT

```

```

62.          THIS IS AUXILIARY FILE DATA
63.          % SET NORMAL-OUTPUT

```

Output source:

```

001700 IDENTIFICATION DIVISION.                                17.
001800 PROGRAM-ID.                                             18.
001900                                     EXAMPLE9.             19.
002000 ENVIRONMENT DIVISION.                                    20.
002100 INPUT-OUTPUT SECTION.                                    21.
002200 FILE-CONTROL.                                          22.
002300     SELECT YOUR-FILE          ASSIGN UT-S-YOURFILE.     23.
005300     SELECT MY-FILE            ASSIGN UT-S-MYFILE.       53.
005302 DATA DIVISION.                                         24.
005304* PLACING DATA IN DATA DIVISION                       46.
005306 FILE SECTION.                                           25.
005308 FD  YOUR-FILE.                                           26.
005310 01  YOUR-RECORD          PIC X(80).                     27.
005312 FD  MY-FILE.                                               49.
005314 01  MY-RECORD          PIC X(80).                     50.
005316 WORKING-STORAGE SECTION.                                  28.
005318 01  COUNTER              PIC S9(4) COMP.                29.
005320 01  WS-1                 PIC X.                         30.
005322 01  TEST-NO-DELIMITERS   PIC 9(6).                     31.
005324*   PLACE FOR STUFF THAT MUST BE AT END OF WS FOR      32.
005326*   CERTAIN NON-APS PRE-PROCESSORS.                     33.
005328 LINKAGE SECTION.                                         34.
005330 01  LINKAGE-STUFF        PIC X.                         40.
006300 01  MY-EPILOGUE-CHR      PIC X.                         41.
006302 PROCEDURE DIVISION.                                       34.
006304     COUNTER = 1                                           35.
006306     COUNTER = 2
006308     COUNTER = 3
006310     TEST-NO-DELIMITERS = 3
006312     COUNTER = 4
006314     COUNTER = 5
006316     COUNTER = COUNTER + 4 5
006318     COUNTER = TEST-NO-DELIMITERS
006320     COUNTER = 6
006322     OPEN INPUT MY-FILE YOUR-FILE                          55.
006324     CLOSE MY-FILE YOUR-FILE                               56.
006326*/ * THIS IS AN S-COBOL COMMENT LINE.                    60.
006328     COUNTER = 100                                         11.
006330     COUNTER = 101                                         15.

```

Discussion

- Lines 1-14, Macros. Three macros are defined: \$IN-LINE, \$MY-EPILOGUE, and \$SECOND-EPILOGUE.
- Line 15, Macros. % SET EPILOGUE \$MY-EPILOGUE causes \$MY-EPILOGUE to be automatically invoked at the very end of the program (after line 63).
- Line 31, Input. A marker for the code on line 58 that is relocated to WORKING-STORAGE; it will go immediately above this marker.
- Line 36, Input. \$IN-LINE is invoked, passing the actual argument 2 to the macro formal argument &ARG. The result appears in output line 6306 (COUNTER = 2).
- Line 37, Input. First, the contents of the evaluation brackets are evaluated. The brackets contain a call to the \$IN-LINE macro, passing the argument 3 to the macro argument &ARG. Thus the macro is called and the result appears in output line 6308 (COUNTER = 3). Next, the contents of the brackets are resolved, giving TEST-NO-DELIMITERS a value. The value comes from a variable within \$IN-LINE that exists for this purpose; this variable must have same name as the macro it belongs to, and its value is the same as the argument passed to the macro (for example, 3). Thus <\$IN-LINE(3)> is replaced by 3, and TEST-NO-DELIMITERS = 3 is output to line 6310.
- Line 38, Input. Outputs lines 6312-6316. If we had wanted to add 45 to COUNTER instead of 4 and 5, we would have omitted the space between the first and second appearance of the in-line macro.
- Lines 39-41, Input. Line 39 generates the LINKAGE SECTION statement and relocates line 40 to the Linkage Section (see output lines 5328-5330); line 41 ends the effect of % SET LINKAGE.
- Line 42, Input. Lets us omit delimiters around string arguments, as on line 43.
- Line 45-47, Input. Line 45 relocates line 46 to immediately after the DATA DIVISION statement (see output line 5304); line 47 ends the effect of % SET DATA.
- Line 48-51, Input. Line 48 relocates lines 49-50 to the end of the File Section (see output lines 5312-5314); line 51 ends the effect of % SET FILE.

- Lines 52-54, Input. Line 52 relocates line 53 to the end of FILE-CONTROL.; line 54 ends the effect of % SET FILE-CONTROL.
- Lines 57-59, Input. Line 57 relocates line 58 to the end of WORKING-STORAGE; line 59 ends the effect of % SET WORKING-STORAGE.
- Lines 61-63, Input. Valid only for APS for z/OS product; line 61 causes line 62 to be output to an external auxiliary file named MACRO; line 63 ends the effect of % SET AUXILIARY-OUTPUT.
- Line 6328, Output. Line 15 of the macro source (% SET EPILOGUE \$MY-EPILOGUE) put \$MY-EPILOGUE in the storage area called EPILOGUE. All macros in EPILOGUE are automatically invoked at the end of the program, after all other all Customization Facility processing is completed, starting with the last macro put into EPILOGUE. Because \$MY-EPILOGUE is the only macro in EPILOGUE, it is invoked. The result is line 6328 in the output.

During processing of \$MY-EPILOGUE, % SET LINKAGE relocates line 9 to the end of the Linkage Section (line 6300), and puts COUNTER = 100 at the end of the Procedure Division (line 6328). Line 15 is an EPILOGUE statement nested within a \$MY-EPILOGUE; it puts \$SECOND-EPILOGUE into the EPILOGUE area. The processing of \$MY-EPILOGUE is now completed.

Now that \$MY-EPILOGUE is finished and \$SECOND-EPILOGUE has become the last macro existing in EPILOGUE, \$SECOND-EPILOGUE is automatically invoked.

After the above program is processed by the Customization Facility, it is processed by the APS Precompiler. The result is shown below. Note the additional lines (5324 through 5340 and 6236 through 6242) that have been generated by the Precompile process.

After APS Precompiler processing:

```

001700 IDENTIFICATION DIVISION.                17.
001800 PROGRAM-ID.                               18.
001900                                     EXAMPLE9.  19.
002000 ENVIRONMENT DIVISION.                    20.
002100 INPUT-OUTPUT SECTION.                   21.
002200 FILE-CONTROL.                           22.
002300     SELECT YOUR-FILE                     ASSIGN UT-S-YOURFILE.  23.
005300     SELECT MY-FILE                       ASSIGN UT-S-MYFILE.  53.

```

005302	DATA DIVISION.		24.
005304*	PLACING DATA IN DATA DIVISION		46.
005306	FILE SECTION.		25.
005308	FD YOUR-FILE.		26.
005310	01 YOUR-RECORD	PIC X(80).	27.
005312	FD MY-FILE.		49.
005314	01 MY-RECORD	PIC X(80).	50.
005316	WORKING-STORAGE SECTION.		28.
005318	77 COUNTER	PIC S9(4) COMP.	29.
005320	01 WS-1	PIC X.	30.
005322	01 TEST-NO-DELIMITERS	PIC 9(6).	
005322			
005324	01 GENERATED--FLAGS.		
005326	05 TRUE	PIC X VALUE 'T'.	
005328	88 ALWAYS	VALUE 'T'.	
005330	88 NEVER	VALUE 'F'.	
005332	05 FALSE	PIC X VALUE 'F'.	
005334	05 YOUR-FILE--END	PIC X.	
005336	05 YOUR-FILE--INV	PIC X.	
005338	05 MY-FILE--END	PIC X.	
005340	05 MY-FILE--INV	PIC X.	
005342			
005344*	PLACE FOR STUFF THAT MUST BE AT END OF WS FOR		32.
005346*	CERTAIN NON-APS PRE-PROCESSORS.		33.
005348	LINKAGE SECTION.		
005350	01 LINKAGE-STUFF	PIC X.	40.
006200	01 MY-EPILOGUE-CHR	PIC X.	
006202	PROCEDURE DIVISION.		34.
006204	MAIN--SECTION SECTION.		35.
006206	MAIN--SECTION--PARA.		35.
006208	COUNTER = 1.		35.
006210	COUNTER = 2.		
006212	COUNTER = 3.		
006214	TEST-NO-DELIMITERS = 3.		
006216	COUNTER = 4.		
006218	COUNTER = 5.		
006220	COUNTER = COUNTER + 4 5		
006222	COUNTER = TEST-NO-DELIMITERS.		
006224	COUNTER = 6.		
006226	OPEN INPUT MY-FILE YOUR-FILE.		55.
006228	CLOSE MY-FILE YOUR-FILE.		56.
006230*	THIS IN AN S-COBOL COMMENT LINE		60.
006232	COUNTER = 100.		11.
006234	COUNTER = 101.		15.
006236	MAIN--SECTION--EXIT.		15.
006238	EXIT PROGRAM.		15.
006240	MAIN--SECTION--EXIT.		15.
006242	STOP RUN.		15.

4 Administration

The Customization Facility Control System lets APS administrators regulate developer access to user-defined Customization Facility macros. Use it to do the following:

- Restrict all developers from using any user-defined macros (default)
- Allow all developers to use all user-defined macros
- Specify selected user-defined macros that all developers can use by creating a rule list, which is a list of those macros or macro libraries.

To create a rule list, follow these steps:

- 1 Starting on the APS Main Menu, select the following Actions to access the Customization Control Menu:
 - Utilities
 - Custom Utilities
 - Admin and Config Facility
 - Customization Facility Control System
- 2 On the Customization Control Menu, select Enable/Disable Customization Control. The Customization Control Activation window displays.
- 3 Complete the window as follows:

Field	Description and Values
Password	Enter the Customization Facility Control System password supplied with your APS software.
Change Password	Lets you change the password.
New Password	If you select the Change Password option, enter a new password in this field.
Display Settings	Displays whether the control system is on or off.

Field	Description and Values
Restrict All Macro Usage	Default. Restricts developers from using any user-defined Customization Facility macros.
Enable Macro Usage	Allows developers to use all user-defined Customization Facility macros or those macros that you specify in a rule list.

- In the USERMACS PDS or data set, create a control file and specify in it the names of macros that developers can use or create. To enable the use of all macros in a specific macro library, specify the macro library name. You will use this control file to generate the rule list in step 7. For example, to enable the use of the macros `macro1` and `macro2`, and all macros in the macro libraries `maclibx` and `macliby`, write the following statements:

```
//INPUT project1.group1.USERMACS
$macro1
$macro2
maclibx
macliby
```

- Redisplay the Customization Control Menu as described in step 1.
- On the Customization Control Menu, select Actions, Rule List Maintenance. The Rule List Maintenance window displays.
- Complete the window as follows:

Field	Description
Password	Enter your Customization Facility password.
Control File	Specify the fully qualified name of the control file that you created in step 4.
Output File	Specify the fully qualified rule list name. The rule list must be named <code>MACROLST</code> and reside in the APS CNTL PDS or data set. For example, SYS1.APS5000.CNTL.MACROLST
Options	V Generates a report of macros in the rule list.

- Select Build New Rule List to generate the rule list.

Index

Symbols

- \$ Customization Facility symbol 7
- % Customization Facility symbol 7
- & Customization Facility symbol 7, 28
- + column indicator 29
- + variable/suffix concatenator 31

A

- arguments
 - in macro calls 7, 39
 - in macro definitions 7, 22
- auxiliary evaluation brackets 47

B

- BEGIN Customization Facility statement 15
- BLANK Customization Facility statement 46
- blank lines
 - in Customization Facility macros 46
- brackets, evaluation
 - setting 47
 - using 25

C

- calling Customization Facility macros 39
- column indicator Customization Facility
 - function 29
- Comments
 - in Customization Facility macros 16
- COMMUNICATION SECTION Customization
 - Facility statement 46
- COMPILETIME Customization Facility func-
 - tion 29
- concatenating
 - variable and suffix 31
- conditional processing with Customization
 - Facility
 - IF structure 33
 - REPEAT structure 42
 - UNTIL structure 42, 51
 - WHILE structure 42, 51
- continuation
 - in Customization Facility macros 17
- CONVERT-LOWER-CASE Customization Facil-
 - ity statement 46
- copylibs/copybooks
 - including in program 35, 49
- Customization Control Activation Window
 - 73
- Customization Facility macros
 - APS and user-defined 5
 - arguments in macro calls 39
 - arguments in macro definitions 22
 - blank lines 46
 - calling 39
 - Comments 16
 - conditional processing 33, 42, 51

- continuing statements 17
- DECLARE structures 18
- definition statement 22
- error handling 23, 46, 48
- escaping from 24
- evaluation brackets 25
- examples 9, 55
- functions 28
- IF structures 33
- including macro libraries 35, 49
- indentation 15
- limits 36
- LOOKUP structures 37
- looping 42, 51
- overview 5
- positioning output 15
- program locations, specifying for source 49
- SET statements 49
- SY* keywords 49
- symbols, reserved 7
- trace facility 23
- variables, assigning values 53

D

- DATA DIVISION Customization Facility statement 46
- debugging programs
 - Customization Facility source 23
- DECLARE Customization Facility structure 18
- DEFINE Customization Facility statement 22
- DEFINED Customization Facility function 29
- DEFVAL Customization Facility function 29
- DELIMITERS Customization Facility statement 46

E

- END-WORKING-STORAGE Customization Facility statement 46
- EPILOGUE Customization Facility function 28
- EPILOGUE Customization Facility statement 46
- ERROR Customization Facility statement 23, 46
- error handling
 - Customization Facility 23, 46, 48
- ESCAPE Customization Facility statement 24
- Eval-BRACKETS Customization Facility statement 47
- evaluation brackets
 - setting 47
 - using 25

F

- FATAL Customization Facility statement 23, 46
- File Section
 - SET FILE SECTION statement 47
- FILE-CONTROL Customization Facility statement 47
- flags
 - loop counters, LOOP-LIMIT 44, 48, 52
- FULL word Customization Facility function 28
- functions
 - Customization Facility 28

H

- HALF word Customization Facility function 29

I

- IF Customization Facility structure 33
 - including in programs
 - copylibs/copybooks 35, 49
 - macro libraries 35, 49
- INDENT Customization Facility function 29
- indentation
 - Customization Facility structures 15
- INDEX Customization Facility function 29
- INFO Customization Facility statement 23, 46

L

- LENGTH Customization Facility function 29
- limits, Customization Facility 36
- Linkage Section
 - SET LINKAGE SECTION statement 47
- literals, Customization Facility
 - appending to variables 31
 - as macro call arguments 39
 - concatenating 54
 - continuing in statements 17
 - delimiters 46
 - parsing 30
- locations, program
 - specifying for Customization Facility
 - source 49
- LOOKUP Customization Facility structure 37
- looping
 - loop counters, LOOP-LIMIT 44, 48, 52
 - REPEAT Customization Facility statement 42
 - SET LOOP-LIMIT Customization Facility statement 48
 - UNTIL Customization Facility statement 42, 51
 - WHILE Customization Facility statement 42, 51

- LOOP-LIMIT Customization Facility statement 48
- LOOP-LIMIT flag 44, 48, 52

M

- macros, user-defined
 - see Customization Facility macros
- MARGIN Customization Facility statement 47
- margins
 - setting for Customization Facility processing 47

N

- nested IF statement 34
- NOBLANK Customization Facility statement 46
- NOTRACE Customization Facility statement 48
- NOWRITE-CONTROL Customization Facility statement 48
- NUMERIC Customization Facility function 29

P

- PARSE Customization Facility function 30
- plus symbol
 - as column indicator 29
 - as variable/suffix concatenator 31
- PRESERVE-LOWER- CASE Customization Facility statement 46
- Procedure Division
 - SET PROCEDURE statement 48
- program locations
 - specifying for Customization Facility
 - source 49

PROGRAM-ID Customization Facility function 29
 PSB-NAME Customization Facility function 29

R

REPEAT Customization Facility statement 42
 REPEAT VARYING Customization Facility statement 42
 Rule List Maintenance window 74

S

SET BLANK Customization Facility statement 46
 SET COMMUNICATION SECTION Customization Facility statement 46
 SET CONVERT-LOWER-CASE Customization Facility statement 46
 SET DATA DIVISION Customization Facility statement 46
 SET DELIMITERS Customization Facility statement 46
 SET END-WORKING-STORAGE Customization Facility statement 46
 SET EPILOGUE Customization Facility statement 46
 SET ERROR Customization Facility statement 23, 46
 SET EVAL-BRACKETS Customization Facility statement 47
 SET FATAL Customization Facility statement 23, 46
 SET FILE SECTION Customization Facility statement 47
 SET FILE-CONTROL Customization Facility statement 47
 SET INFO Customization Facility statement 23, 46

SET LINKAGE SECTION Customization Facility statement 47
 SET LOOP-LIMIT Customization Facility statement 48
 SET MARGIN Customization Facility statement 47
 SET NOBLANK Customization Facility statement 46
 SET NOTRACE Customization Facility statement 48
 SET NOWRITE-CONTROL Customization Facility statement 48
 SET PRESERVE-LOWER-CASE Customization Facility statement 46
 SET PROCEDURE Customization Facility statement 48
 SET SPECIAL-NAMES Customization Facility statement 48
 SET TRACE ERROR Customization Facility statement 23, 48
 SET WARNING Customization Facility statement 23, 46
 SET WORKING-STORAGE Customization Facility statement 48
 SET WRITE-CONTROL Customization Facility statement 24, 48
 size limitations in APS 36
 SPECIAL-NAMES Customization Facility statement 48
 SUBSTR Customization Facility function 30, 32
 symbols
 Customization Facility reserved 7

T

TRACE ERROR Customization Facility statement 23, 48
 trace facility
 Customization Facility 23

U

UNTIL Customization Facility statement 42,
51

V

variables, Customization Facility
 assigning values to 53
 concatenating suffix to 31

W

WARNING Customization Facility statement
 23, 46
WHILE Customization Facility statement 42,
 51
windows, APS
 Customization Control Activation win-
 dow 73
 Rule List Maintenance window 74
WORKING-STORAGE Customization Facility
 statement 48
WRITE-CONTROL Customization Facility
 statement 24, 48

