

# Dynamic JCL Generation (Job Type MAC)

This subsection covers the following topics:

- What Is Dynamic JCL Generation?
  - Editing and Generation of MAC Jobs
  - Escape Character
  - Variables in Dynamically Generated JCL
  - Symbols and Local Variables
  - Inserting Text Modules into the JCL
  - Parameters for Included Text Modules
  - Nested (Recursive) #EOR-INCL Statements
  - Replacement of Parameters within the Text Module
  - Examples
- 

## What Is Dynamic JCL Generation?

When Entire Operations activates a job network, the JCL of the jobs in the network is copied onto the active data base. Entire Operations provides a facility which allows you to use variables in the original JCL and which can create parts of the JCL depending on program logic. Variables are substituted by their current values either at activation time or at job submission time (see the subsection Symbol Replacement in this section). This is referred to as Dynamic JCL Generation and applies only to MAC-type jobs in Entire Operations.

Dynamically generated JCL is useful if you wish the JCL to contain a step only under certain circumstances, for example, if the current date is YYYYMMDD, include job step X.

Dynamic JCL can be defined for MAC-type jobs using the Edit function in the Job Maintenance facility of Entire Operations. To convert existing JCL to the Entire Operations MAC format, use the JCL-IMPORT function in the job definition with JCL location as NAT. In all cases, the Editor command MACRO must be used to generate the final JCL. The Editor command TEST can be used to test the generated JCL.

## Editing and Generation of MAC Jobs

For the editing and creation of MAC jobs, you must use the built-in SAG Editor.

For further information on special editor commands for MAC jobs, please see Editing JCL of MAC (Macro) Jobs.

## Escape Character

The activation escape character at the beginning of a line distinguishes the line as a Natural statement from the JCL. The use of Natural statements provides full Natural functionality in dynamic JCL, including access to Adabas and Entire System Server. This means you do not need to learn any special control statements. All Natural statements used in dynamic JCL must be coded in structured mode.

Variables are user-defined and can be used in any part of the JCL, including the file name and control card. Variables are distinguished in the JCL by preceding them with an escape character: the **activation escape** denotes variables to be substituted at activation time, the **submission escape** denotes variables to be substituted at job submission time.

### Note:

These escape characters can be changed by the system administrator. However, this should only be done if absolutely necessary, for example for countries with a different alphabet. See the subsection Symbol Escape Characters above.

It is not advisable to use escape characters which have a defined meaning in a particular operating system or which are already defined as escape characters, e.g. \$ in BS2000/OSD or UNIX.

## Variables in Dynamically Generated JCL

- Sample JCL
- Symbol Table Variables
- Natural System Variables
- Variables from the Parameter Section

You can use four types of variables:

- A variable from the symbol table specified for the job;
- A variable from the parameter section (P-...);
- A local variable defined in this Natural program, which can be computed in your program (L-...);
- A Natural system variable (TIME, DATE etc.) which is distinguished by an asterisk (\*), for example, \*TIME.

### Note:

All variables with prefixes other than 'P-...' 'L-...' or '\*...' are assumed to be found in the symbol table.

Symbol replacement itself (without embedding Natural statements) is also available for standard JCL (JOB-type jobs). See the subsection Symbol Replacement in this section.

## Sample JCL

The following sample JCL illustrates the use of variables in the dynamically generated JCL of a MAC-type job in Entire Operations:

```

§ DEFINE DATA PARAMETER USING NOXPPL-A
§ LOCAL      /* ... ALL LOCALS SHOULD START WITH 'L-'
§ 1 L-01     (A30)
§ 1 CLASS   (A01)      /* FROM SYMBOL TABLE, FOR #GET-SYMBOL
§ END-DEFINE
§ * -----
#GET-SYMBOL CLASS
§ COMPRESS P-NETWORK P-JOB INTO L-01
//SNNOPLEX JOB ,§P-OWNER,MSGCLASS=§MSGCLASS,CLASS=§CLASS
//STEP01 EXEC PGM=NOPCONTI,PARM='C=0004'
//STEPLIB DD DISP=SHR,DSN=§STEPLIB
/* DEVICE: §*DEVICE, INIT-USER: §*INIT-USER, TIME: §*TIME
/* L-01 : §L-01
§ IF CLASS = 'K'      /* SYMBOL USED IN STATEMENT -> #GET-SYMBOL
/* THE CLASS IS §CLASS
§ ELSE
/* ANOTHER MSGCLASS FOUND
§ END-IF
/*

```

The variables used in the JCL are assumed to have the following current values:

## Symbol Table Variables

<b>STEPLIB</b>	NOP.SYSF.DEV.LOAD
<b>CLASS</b>	K
<b>MSGCLASS</b>	X

## Natural System Variables

<b>*DEVICE</b>	BATCH
<b>*INIT-USER</b>	EORMON

## Variables from the Parameter Section

<b>P-NETWORK</b>	EX131A
<b>P-JOB</b>	EX-1-24
<b>P-OWNER</b>	SN

When the job is activated, Entire Operations substitutes the variables with their current values. The following JCL is generated as a result:

```
//SNNPEX JOB ,SN,MSGCLASS=X,CLASS=K
//STEP01 EXEC PGM=NOPCONTI,PARM='C0004'
//STEPLIB DD DISP=SHR,DSN=NOP.SYSF.DEV.LOAD
//* DEVICE: BATCH, INIT-USER: EORMON
//* L-01 : EX131-A EX-1-24
//* THE CLASS IS K
/*
```

See also the section Job Maintenance.

## Symbols and Local Variables

- #GET-SYMBOL
- #SET-SYMBOL
- #SET-SYMBOL-M
- General Notes
- Examples

To copy symbol values from the (active) symbol table to a local variable in the Macro program and vice versa, the built-in statements #GET-SYMBOL and #SET-SYMBOL can be used.

### #GET-SYMBOL

#GET-SYMBOL can be used within a MACRO JCL to put an active symbol value into a local variable.

The symbol is taken from where it is found first in the symbol search path.

#### Syntax

```
#GET-SYMBOL <variable> <value>
```

<value> is optional. If it is not specified, the variable will be filled with the contents of a symbol with the same name.

## #SET-SYMBOL

#SET-SYMBOL can be used within MACRO JCL to set an **active** symbol and its value from a local variable or string.

The symbol will be set in the **active** symbol table which belongs to the active job.

### Syntax

```
#SET-SYMBOL <symbol> <value>
```

<value> is optional. If it is not specified, the symbol will be filled with the contents of a local variable with the same name.

## #SET-SYMBOL-M

#SET-SYMBOL can be used within MACRO JCL to set a **master** symbol and its value from a local variable or string.

The symbol will be set in the **master** symbol table which belongs to the active job.

### Syntax

```
#SET-SYMBOL-M <symbol> <value>
```

<value> is optional. If it is not specified, the symbol will be filled with the contents of a local variable with the same name.

## General Notes

Text strings must not contain blanks.

Blanks are used as separators for the parameters of #GET-SYMBOL and #SET-SYMBOL.

## Examples

(The activation escape character is §.)

Statement	Description
#GET-SYMBOL J	Move the contents of symbol J to the local variable J.
#GET-SYMBOL #J §BB	Move the contents of symbol BB to the local variable #J.
#GET-SYMBOL #J '§BB'	Move the contents of symbol BB to the local variable #J.
#GET-SYMBOL L-MULT '§?MV<M1,§I>'	Move the result of a symbol function call into the local variable L-MULT.
#SET-SYMBOL J	Set the <b>active</b> symbol J to the contents of the local variable J.
#SET-SYMBOL I #A	Set the <b>active</b> symbol I to the contents of the local variable #A.
#SET-SYMBOL AA 'text'	Set the <b>active</b> symbol AA to the value 'text'.
#SET-SYMBOL BB §I	Set the <b>active</b> symbol BB to the contents of symbol I.
#SET-SYMBOL BB '§I'	Set the <b>active</b> symbol BB to the contents of symbol I.
#SET-SYMBOL CC '§D.§E'	Concatenate symbol values of D and E, and put the result into the <b>active</b> symbol CC.
#SET-SYMBOL-M DD ‡valueŸ	Set the <b>master</b> symbol DD to the value 'valueŸ'.

## Inserting Text Modules into the JCL

You can insert Natural text modules anywhere in your JCL.

This feature is not limited to jobs of type MAC (Macro), but can also be used from within standard JCL.

The syntax is as follows:

```
#EOR-INCL LOC=NAT LIB=<library> MEM=<member>
```

### Parameters of the #EOR-INCL Statement

Parameter	Description
LOC	Location NAT Natural text member This is the only location which is currently allowed for text modules.
LIB	Library of the text module
MEM	Name of the text module

If the text module cannot be read, the JCL generation will be aborted with an error message.

Symbol replacement is possible within the #EOR-INCL statement.

#### Note:

Before Entire Operations 4.1.1, the statement #INCLUDE was available for inclusion. This is still supported in Entire Operations 4.1.1, but will not be supported in future. For #INCLUDE, the following features will not be supported. It is recommended to convert #INCLUDE statements into #EOR-INCL statements with Entire Operations 4.1.1.

## Parameters for Included Text Modules

You may invoke included text modules with specific parameters.

For each parameter you want to pass to the text which is included by an #EOR-INCL statement, you must code an #EOR-PARM line in front of the #EOR-INCL statement.

These parameters will be in effect only for the text included (and all nested inclusions) in the following #EOR-INCL statement.

The syntax is as follows:

```
#EOR-PARM <parameter> = <value> [ <parameter> = <value> ]
```

Multiple parameters can be passed on the #EOR-PARM line. This is limited by the line size.

Parameter values implicitly have the format A (alphanumeric).

Parameter values may contain blanks. In this case, they must be included in apostrophes (') or double quotes (").

## Nested (Recursive) #EOR-INCL Statements

You may use nested #EOR-INCL statements inside modules, which have already been included. These nested inclusions may have their own parameter lists.

### Note:

Be aware that the total number of inclusions for a JCL is limited by the Natural Editor buffer size for the Natural task executing.

## Replacement of Parameters within the Text Module

Parameters for text modules can be used within the included text module like any other symbol. They have precedence over the symbols of the current symbol table of the job (and the other symbol tables in the search hierarchy).

Therefore, parameters for included text members can temporarily override symbols with the same name. The scope of a parameter is only the included text module, and the text modules which are invoked from within this text module by nested calls.

The lifetime of parameters defined via #EOR-PARM is JCL load time only. Normal symbol replacement can be used within included text modules like everywhere else in the JCL.

## Examples

### Example 1

```
#EOR-INCL LOC=NAT LIB=JCLLIB MEM=$MEMBER
```

Include the Natural member, the name of which is in the symbol MEMBER, from the library JCLLIB.

## Example 2

```
#EOR-PARM DBID = 9
#EOR-PARM TEXT-1 = a string with blanks
#EOR-INCL LOC=NAT LIB=USERLIB MEMBER=BLOCK1
/* JCL statement
#EOR-PARM DBID = 10
#EOR-PARM TEXT-1 = another string
#EOR-INCL LOC=NAT LIB=USERLIB MEMBER=BLOCK1
```

Include the Natural text member **BLOCK1** from the library **USERLIB**. The parameters **DBID** and **TEXT-1** are passed to the text module, with different values for the 2 invokings.