

Dynamic JCL Generation

This subsection covers the following topics:

- General
 - Example 1: Dynamic JCL in an OS/390 Environment
 - Example 2: Dynamic JCL in a BS2000/OSD Environment
 - Example 3: Dynamic JCL in a UNIX Environment
-

General

When defining a job within a network, a user can specify that its JCL is to be generated dynamically either at job activation time or at job submission time.

Dynamic JCL generation is achieved using the Entire Operations MACRO facility, an extension of the Natural programming language. This facility consists of standard Natural statements and text strings (JCL frames). The text strings can contain Natural escape characters followed by variables that will be replaced by their current value during dynamic generation.

These current values will be taken from the so called "Symbol Tables", which are user-defined tables defining the escape characters and variable names used in the MACRO facility, as well as the current values to be substituted. The symbol table to be used is specified in the job definition screen.

If any symbol specified in the dynamic JCL is not in the symbol table indicated for the job, the symbol is searched for at substitution time (either activation or execution) in the symbol table(s) belonging to owner SYSDBA. A user can define any number of entries in a single symbol table or any number of symbol tables.

Additionally, Entire Operations passes standard variables defined in the parameter section to the dynamically generated program, such as job owner, network name, current job name and original scheduling date. The same applies to Natural system variables such as DATE, TIME and USER. As these parameters can be replaced in any part of the JCL, different JCL configurations can be generated depending on time, date, user ID etc.

For more information concerning the editing of jobs of type MAC, see the subsection Editing JCL of MAC (Macro) Jobs.

Entire Operations provides dynamic JCL generation for all supported platforms (OS/390, VSE/ESA, BS2000/OSD, UNIX) as shown in the following examples:

- Example 1: Dynamic JCL in an OS/390 Environment
- Example 2: Dynamic JCL in a BS2000/OSD Environment
- Example 3: Dynamic JCL in a UNIX Environment

Example 1: Dynamic JCL in an OS/390 Environment

The following is the symbol table specified for the MACRO program:

Symbol Name	Current Value
STEPLIB	SN.SYSF.SOURCE
CLASS	G

The variable from the parameter section is assumed to have the following value:

P-OWNER	NET1
---------	------

The system variables are assumed to have the following values:

*TPSYS	COMPLETE
*DEVICE	BATCH
*INIT-USER	SN

The following is a Natural MACRO program including a parameter section and JCL with the Natural escape character (paragraph sign #) followed by variable names from the symbol table.

```
# DEFINE DATA PARAMETER USING NOPXPL-A
# LOCAL /* MUST BE CODED
# END-DEFINE
//SNMAC4 JOB ,#P-OWNER,MSGCLASS=X,CLASS=#CLASS //STEP01 EXEC
PGM=NOPCONTI,PARM='C0004' //STEPLIB DD DISP=SHR,DSN=#STEPLIB
/* DEVICE: *DEVICE, INIT-USER: *INIT-USER /* TPSYS: *TPSYS
# IF CLASS = 'G'
/* THE MSGCLASS IS REALLY 'G'
# ELSE
/* ANOTHER MSG-CLASS FOUND
# END-IF
/*
```

The resulting dynamically generated JCL will be:

```
//SNMAC4 JOB ,NET1,MSGCLASS=X,CLASS=G
//STEP01 EXEC PGM=NOPCONTI,PARM='C0004' //STEPLIB DD
DISP=SHR,DSN=SN.SYSF.SOURCE /* DEVICE: BATCH, INIT-USER: SN
/* TPSYS: COMPLETE
/* THE MSGCLASS IS REALLY 'G'
/*
```

Example 2: Dynamic JCL in a BS2000/OSD Environment

The fields taken from the DB-INFO are assumed to have the following values after the FIND statement:

Field	Value
NUCLEUS	055
LP1	1000
NU1	100
ACCOUNT	EXAMPLE
NH1	4000
MSG	FHL
VERSION	524

The variables taken from the parameter section have the following current values:

Variable	Value
P-OWNER	OS
P-JOB	NUC055
P-EXECUTION-NODE	055

No symbol table was defined for this example job.

The following is the example JCL written using the Natural MACRO facility, including variables to be substituted from the DB-INFO view and the parameter section. Variables are preceded by the escape character paragraph sign #.

```
# DEFINE DATA PARAMETER USING NOPXPL-A
# 1 L-JOB
# 1 REDEFINE L-JOB
# 2 L-JOB-A (A3)
# 2 L-JOB-NUC (N3)
# LOCAL /* LOCAL VARIABLES START HERE
# 1 DB-INFO VIEW OF DB-INFO
# 2 NUCLEUS
# 2 LP1
# 2 NU1
# 2 ACCOUNT
# 2 NH1
# 2 MSG
# 2 VERSION /* E.G. 524
# 1 LWP (N7)
# 1 NUC (N3)
# 1 SPOOL (A10) INIT <'NOSPOOL'>
# END-DEFINE
# *
# MOVE P-JOB TO L-JOB-A
# MOVE P-EXECUTION-NODE TO NUC
# F1. FIND DB-INFO WITH NUCLEUS = NUC
/.NUC NUC LOGON #P-OWNER,#ACCOUNT
/OPTION MSG=#MSG
/REMARK
/REMARK NUCLEUS #NUC
/REMARK
/SYSFILE SYSLST = NUC NUC..LST.NUC
/SYSFILE SYSDTA = SYSCMD
/FILE ADA VERSION..MOD,LINK=DDLIB
/FILE *DUMMY,LINK=DDLOG
/FILE *DUMMY,LINK=DDSIBA
/FILE ADA NUC..ASSO,LINK=DDASSOR1,SHARUPD=YES
/FILE ADA NUC..DATA,LINK=DDDATAR1,SHARUPD=YES
/FILE ADA NUC..WORK,LINK=DDWORKR1,SHARUPD=YES
/EXEC (ADARUN,ADA VERSION..MOD)
# COMPUTE LWP = F1.LP1 * (F1.NU1 + 100)
ADARUN PROG=ADANUC,LP=F1.LP1,LU=65535,LWP=#LWP ADARUN
DB=#NUC,NU=#NU1,NC=20,TT=600,TNAE=1800 ADARUN NH= NH1
/SYSFILE SYSLST = (PRIMARY)
/SYSFILE SYSDTA = (PRIMARY)
/SYSFILE SYSOUT = (PRIMARY)
/LOGOFF SPOOL
# END-FIND
```

The resulting dynamically generated JCL will be:

```

/.NUC055 LOGON OS,EXAMPLE
/OPTION MSG=FHL
/REMARK
/REMARK  NUCLEUS 055
/REMARK
/SYSFILE  SYSLST = NUC055.LST.NUC
/SYSFILE  SYSDTA = SYSCMD
/FILE  ADA524.MOD,LINK=DDLIB
/FILE  *DUMMY,LINK=DDLOG
/FILE  *DUMMY,LINK=DDSIBA
/FILE  ADA055.ASSO,LINK=DDASSOR1,SHARUPD=YES
/FILE  ADA055.DATA,LINK=DDDATAR1,SHARUPD=YES
/FILE  ADA055.WORK,LINK=DDWORKR1,SHARUPD=YES
/EXEC  (ADARUN,ADA524.MOD)
ADARUN  PROG=ADANUC,LP=1000,LU=65535,LWP=200000 ADARUN
DB=055,NU=100,NC=20,TT=600,TNAE=1800 ADARUN NH=4000
/SYSFILE  SYSLST = (PRIMARY)
/SYSFILE  SYSDTA = (PRIMARY)
/SYSFILE  SYSOUT = (PRIMARY)
/LOGOFF  NOSPOOL
    
```

Note:

Any JCL generated at activation time using the MACRO language can be modified by the user until the job is actually submitted. Of course this modification is valid only for the current network run.

Example 3: Dynamic JCL in a UNIX Environment

The following example illustrates dynamic symbol replacement within a Bourne shell script (escape character \$):

```

#
# Bourne shell script for checking the number of users
# entered in /etc/passwd.
# If more than $USER-LIMIT entries appear,
# the script will be ended with exit 1.
#
#!/bin/sh
set -x
USER_COUNT='wc -l < /etc/passwd'
echo Number of users on node 'hostname' : $USER_COUNT
if test $USER_COUNT -gt $USER-LIMIT
then
    echo USER_COUNT_WARN
    exit 1
else
    echo USER_COUNT_OK
fi
    
```

The symbol table to be used should appear as follows:

Symbol Name	Current Value
USER-LIMIT	100

The result is the following executable shell script:

```
#
# Bourne shell script for checking the number of users
# entered in /etc/passwd.
# If more than 100 entries appear,
# the script will be ended with exit 1.
#
#!/bin/sh
set -x
USER_COUNT='wc -l < /etc/passwd'
echo Number of users on node 'hostname' : $USER_COUNT
if test $USER_COUNT -gt 100
then
    echo USER_COUNT_WARN
    exit 1
else
    echo USER_COUNT_OK
fi
```

Note:

Any JCL generated at activation time using the Natural MACRO language can be modified by the user until the job is actually submitted. Of course this modification is valid only for the current network run.