System Programming Guide

JOL

Universal Command Language

Version 6.0

Clarke Computer Software.

Jol Universal Command Language "*System Programming Guide".*

Six Edition (February, 2011)

This is a major revision of, and obsoletes,
the Fifth Edition of the Jol Command Language
"System Programming Guide" September 1999

Clarke Computer Software,
Unit 50, 45 Riversdale Road,
Hawthorn, Victoria,
Australia 3122.

Telephone **(61)-601-054-155**
Email **clementclarke@ozemail.com.au**

Preface

This book is a guide to programmers installing Jol.

It describes how to tailor Jol to your specific installation. It is divided into sections to familiarise you with the Jol system requirements, installation and default macros, using initiator dedicated data sets, JOL command processor, post installation and writing invoked routines.

Later sections describe how to alter the Jol defaults and command processor, and, in detail, the parameters that may be used with the assembler macros to alter the behaviour of Jol.

This book assumes that the reader has a working knowledge of the Operating System (MVS, MVS/XA or VS1).

This manual is divided into nine sections:

Section 1: Introduction.

Section 2: System Requirements.

Section 3: Installation and Default Macros.

Section 4: Installing the Command Processor.

Section 5: Post Installation.

Section 6: Writing Invoked Routines.

Section 7: Adding the RACF Exit.

Section 8: Jol Assembler Macros.

Appendix A: Description of TOKENS as used in Jol.

SUMMARY OF RECENT IMPORTANT DEVELOPMENTS TO Jol

Over the last few years, Jol has experienced enormous developments and changes that have been geared to increasing its power, flexibility, proficiency and ease of use. For example, Jol is now twice as powerful as when it was initially released, as measured by its proficiency in operations. Jol can now use Dynamic Allocation so that jobs can run under TSO or in Background, with or without the use of JCL.

TSO Support.

- Options to allow the job to execute under TSO or Background, using the same Command Language.
- Coming - Optional use of **ISPF** for Jol Panels.

ALLOCATE Command.

- **ALLOCATE** a data set in the Preprocessor or Macro Phase for Input or Output.

Addition of OPEN, READ and WRITE Instructions.

- **OPEN** a file for input or output.
- **READ** a record into a variable.
- **WRITE** a record from a variable.

Automatic Reset of Relative Generation Numbers

- Relative Generation Numbers are automatically reset for Reruns or Restarts.

Testing if a data set exists.

- **TEXIST** Command allows the testing of the existence of a data set at execution time.

Coming New Installation Procedures

The basic machine-readable material has been revised, and there are new procedures to install. Separate procedures are used for installation in the OS environment (using SMP) and the non-SMP Installation.

ASSOCIATED DOCUMENTATION INCLUDES:

- **Jol Reference Manual**
- **Jol User Guide**
- **Jol Concepts and Facilities Manual**
- **Jol Answers to Questions**

Jol Evaluation Plan and Financial Work Sheets

- **Jol Conversion Utilities**

Jol Planning and Installation Guide
Jol Release and SMP Guide
Jol Program Logic Manual

1 Introduction

1.1.1 Tailoring Jol for Your Installation

Once the Jol Data Sets have been loaded to disk, two assemblies and link edits must be performed to tailor Jol for your installation. Later, you may wish to also alter some of the Jol *Macro* Commands, and install *Invoke* routines and *User Exits.*

As delivered, the Jol compiler and command processor have defaults that should enable you to execute Jol immediately.

However, the command processor, in particular, requires certain named data sets to execute. To change the defaults and/or the names of the data sets that Jol will use requires two assemblies and link-edits.

Sample assembly and link edit jobs are provided with the tape. You can alter the parameters in these jobs and submit them. The *Installation Guide* contains further details. However, an understanding of the macros and their parameters described in this publication is essential.

1.1.2 The JOLGEN Assembly

The first assembly changes the defaults that the Jol Compiler uses. You also specify to the JOLGEN and associated macros the Compiler Defaults and, optionally, the names of the installation UNITS and VOLUMES.

By specifying to Jol details of particular units, volumes and data sets, better performance of the created job can be expected as Jol can optimize the job globally.

The allowable parameters are described in detail in the section on **JOLGEN**.

1.1.3 The Command Processor Assembly

The second assembly specifies the compiler defaults to be used when Jol is run under TSO. The *default* names of the data sets the command processor will allocate for the user can also be specified in this assembly.

1.1.4 Loading Instructions

Separate loading instructions are provided and dispatched with each tape. Other data sets, such as user contributed Jol macros, are also on the tape. The loading instructions detail how to load Jol, and describe the full contents of the tape.

2 System Requirements

2.1 Mainframe

Jol will run, without modification, on all IBM 370, 303X and 43XX mainframes operating with MFT, MVT, VS1, MVS, MVS/XA with HASP, ASP, JES2 and JES3. It will not run on SVS.

2.1.1 Memory:

Jol requires approximately 200KB Virtual Memory.

2.1.2 Auxiliary Storage:

Permanent: Jol requires approximately 10 cylinders on a 3350 or equivalent device.

Non-Permanent: Jol requires approximately 15 cylinders on a 3350 or equivalent device. These may be held on tape or removable disk.

2.1.3 Jol Data Sets:

Permanent: The **LOAD, COMMAND** and **INCLUDE** data sets are always used by Jol. Therefore, these data sets should be allocated to a permanently mounted device. The data sets are:

SYS2.JOL60.LOAD
SYS2.JOL60.CMDLIB
SYS2.JOL60.INCLUDE

Non-Permanent: The following data sets are used only during the *installation* of Jol or for assemblies of installation exits. Therefore they may be held on tape or demountable disks and loaded when required. The non-permanent data sets are:

SYS2.JOL60.SOURCE
SYS2.JOL60.MACLIB
SYS2.JOL60.PTFS
SYS2.JOL60.PROCLIB
SYS2.JOL60.USER.CMDLIB
SYS2.JOL60.INSTALL

2.1.4 Authorisation and Linklist requirements

The following modules must reside in a **LNKLST** library:-

$JOLIN60
$JOLMN60 and **$JOLSH60**
$JOLCP60

The **$JOLMN60** and **$JOLSH60** modules must be authorized. **$JOLMN60** is the MONITOR program and ATTACHES the problem program. **$JOLSH60** must have access to the System Job Queue, and must be able to use the special **ATTACH** under the operating system to propagate (or cancel) authorization down to the problem program.

2.1.5 Installation Guidelines

Most installations tailor Jol by writing their own Jol Macros and Invoked Routines. Often, the Jol supplied macros are also altered. For example, the Jol **PRINT** command uses a system utility to copy the data set to a

SYSOUT file, but your installation may have a special print utility that it would prefer to use. In this and similar cases, you can alter the Jol macro (with a standard editor such as ISPF) to suit your requirements.

Further considerations are:

- Your installation may use corporate INCLUDE libraries that contain descriptions of all the production data sets and programs in your installation.

- Most Jol Users have their own small Jol data sets which contain private Macros or Jol source code.

To make use of these facilities and to isolate your installation from changes made in new releases of Jol, it is recommended that you place your corporate macros and invoked routines in separate libraries, and have your libraries *placed first in the concatenation list* of data sets required for the Command Processor, that is before the standard Jol supplied macro library. In this way, your macros and invoke routines will be called in preference to those supplied with Jol.

With these points in mind, the Jol Command Processor can be generated with different data sets, or you can generate multiple command processors using different data sets and call them different names.

Therefore the following parameters should be carefully reviewed when generating the Command Processor. They are all described in Section 4 of this manual. The parameters that require special attention are:-

CMDLIB
JLOAD
INCLIB

Other parameters can be used with the default parameters, and changed, if desired, at some later time.

Note particularly: If the Command Processor finds some Jol data sets *already allocated*, it will use those data sets and not the data sets it was generated with. This allows you to have a CLIST that allocates some special data sets for special purposes, but still call the Jol command processor in the usual way.

3 Installation and Default Macros

3.1 JOL Compiler Defaults

The Jol compiler installation macros are discussed in this section. These macros must be coded with standard Assembler macro coding conventions. The Command Processor macro is discussed in a separate section.

The following Jol Compiler Default Option macros are described in this section:-

A) JOLDEF
B) VOLSTART
C) RETVOLS
D) DEFUNIT
E) SWAPUNIT
F) NOCAT

Sample JCL and macro definitions are contained in the Loading Instructions and in the JOLINSTL data set provided.

3.2 The JOLDEF Macro

The **JOLDEF** Macro defines the system type and supplies certain defaults to be applied to the generated code.

Notes

1. Most of these options may be overridden using Compiler Options. These options are fully described in the Jol Reference Manual under the section "Compiler Options".

2. The parameters described in Figure 3-1 may be coded in the JOLDEF macro. For convenience, they are listed in alphabetic sequence.

Note that the shaded items should be carefully examined. Non-shaded items can be changed with the Jol $JOLPREF Macro Command that is automatically executed when Jol starts, or through compiler options.

Parameter	**Default**	**Allowable Values**	**Notes**
CARD1 **CARD2** **CARD3**	*null*	*any valid* **JCL DD** *card*	You may specify up to *three* (3) **DD CARDS** that will be generated each time Jol outputs a new OS job step. Uses include, for example, the updating of a data set showing the current step that has just executed thus making information available for Automatic Restart. The format is: **CARD**$_n$='//*ddname* DD DSN=*name, etc*' For example: **CARD1='//RUNFILE DD DSN=SYS1.RUNFILE, DISP=SHR'** This parameter need not be specified. It can be overridden by JOL Compiler or Command Processor options.
CLASS	*null*	*default-class*	This is the default class to be applied to the generated JOBCARD if **CLASS** is not specified in the Jol job statement supplied by the user. A ***null*** default means that the installation reader default will apply
CPU .	*null*	*default* **CPU-***time (inminutes)*	This is the default amount of CPU time to be used for the generated JOB card if a parameter is not provided by the user on the Jol job statement. If not specified from any source, the installation defaults will apply.
CPUID	**000000**	*6-character-identifier-assigned-to-CPU.*	This is the six position alphanumeric identifier assigned to the installed CPU.
CPUTYPE	**3033**	*7 character CPU name*	Not yet implemented.
DEFDISK	**SYSDA**	*any disk unit name*	**DEFDISK** specifies the default **UNIT** for output data sets if no unit is coded, and if the Data Set is non-_sequential (that is, DSORG=PO, DA, IS or VSAM) or a temporary data set. Note: if the user codes a real data set name and that data set, either implicitly or explicitly, is

Parameter	Default	Allowable Values	Notes
			sequential or defaults to a sequential file, the **DEFTAPE** default (see below) is substituted. If this parameter is nullified i.e. **DEFDISK=,** is specified, and a Jol user attempts to create a new data set without specifying a unit, Jol will (a) notify the user of the omission and (b) prevent the job from executing.
DEFTAPE	**TAPE**	*any sequential unit name*	This specifies the default **UNIT** for *non-temporary sequential* data sets, if one is not specified by the user. Note: The default of SYSDA prevents unnecessary tape loads due to omitting units on new data sets. Installations that have large data sets may prefer to code TAPE, or use the null default, in which case Jol will produce a message indicating that no UNIT was coded. The job will be prevented from running.
DUMP	*null*	*any valid* **SYSUDUMP DD** *statement*	This control statement is used by the **ON ERROR JOL** instruction to set dumps for ABENDS off or on: it is generated for every JOB step. The format is: **DUMP='//SYSUDUMP DD SYSOUT=name, OUTLIM=20000'** If this parameter is null, no automatic dump of a user program will result if an ABEND occurs *unless* altered at compile time by:- 1) an **ON ERROR** statement, 2) using **CARD1**/**CARD2**/**CARD3** (see 1 above) or 3) using the **DC** parameter of the Command Processor.
ELAP	*null*	*elapsed time in minutes*	**ELAP** specifies the default elapsed time to be used if an elapsed time is not specified by the user on a Jol job statement. The elapsed time is used by JES to assist in better scheduling of the computer. As an alternative to using the ELAPSED time parameter (either in the JOLGEN or on the Jol

Parameter	Default	Allowable Values	Notes
			Job Card), it may be useful to write a user validation exit to automatically generate the parameter or to set the JOB CLASS to a specific class for very long jobs or to inform the operator if a long job is received into the system. The null default allows installation defaults to apply.
ERRSEV	**0**	*1 digit number* (In range 0-5)	**ERRSEV** specifies the severity of Jol to be printed. Specification of 0 will result in all the Jol messages being printed. Specification of 5 will result in only terminating error messages being printed. This parameter may be overridden at compile time.
FLAG	**0**	**0** *or* **1-73**	**FLAG** permits an installation to specify an end of column for the Jol source code. For example, supposing that the last column programmers should code in were 71 then coding FLAG=71 will produce an asterisk after column 71 on the printout. This assists programmers to find errors due to coding outside the Source Margins. This parameter is usually omitted.
GDGDCB	**JOLDCB**	*Any valid* DSNAME *to be used as a dcb-reference name for output GDG's*	When a new Generation of a Data Set is created, the Operating System requires that a Cataloged Data Set Name be referenced in the DCB parameter referring to the data set name. Most installations will have a data set used specifically for this purpose. The name of this data set should be coded for this parameter value. This parameter may *not* be nullified.
JOBREGN	*null*	*Default region size*	This is the default region size to be used if a region size is not specified by a user on a Jol job statement. It should be a numeric value. The default region size for **JOBREGN** is null, in which case installation defaults will apply.
JOLCARD	*blanks*	*any two non-blank characters*	When **JOLCARD** is set to any two non-blank characters the first two (2) characters of **ALL** Jol statements are assumed to contain the two characters so specified. Unless there is a particular requirement to differentiate Jol statements from others in the system, it is **strongly advised**, that this field be

Parameter	Default	Allowable Values	Notes
			left blank, thus allowing use of columns 1-71 as Jol source statements. This parameter should not be specified unless it is a particular installation requirement.
LINECNT	**57**	*lines per page of printout*	This indicates the maximum number of print lines per page for the Jol compiler listing. It must be a numeric value. The maximum line count per page permitted in an installation operating environment must be considered when LINECNT is specified. par The maximum line count that may be specified is 32000. This parameter *may not* be nullified.
MAXNAME	**400**	*numeric value*	The **MAXNAME** specifies to Jol the maximum number of DSID's, PGMID's and MACRO names for which Jol is to build tables during compilation. The area of memory required by MAXNAME is the value assigned to MAXNAME times 12 bytes. This parameter may not be nullified.
MAXSYM	**500**	*numeric value*	This parameter specifies the maximum number of symbolics to be allowed for a Jol compilation. The area of memory required by MAXSYM is the value assigned to MAXSYM times 12 bytes. This parameter may not be nullified.
MAXSYMV	**9000**	*numeric value*	The **MAXSYMV** specifies to Jol the amount of virtual storage to be allocated for the values associated with symbolic parameters or variables. This parameter may not be nullified.
MON	**BATCH**	**BATCH** or **DYNAM**	This parameter specifies whether JCL is produced or proper data sets are dynamically allocated by the monitor. This parameter must not be nullified, i.e. do **not** specify **MON=,** This parameter need not be specified. It can altered or specified with options when the compiler is executed.
OUTDEFS	**(,,,5,2,T)**	*any valid sub-*	The sub-parameters specified by OUTDEFS are

Parameter	Default	Allowable Values	Notes
		parameters	default parameters to be used on output data sets. Unless there are specific reasons for an installation to alter the standard installation defaults these should be coded as defaulted. **Sub-parameter Meaning** 1 RECFM 2 BLKSIZE 3 LRECL 4 PRIMARY SPACE 5 SECONDARY SPACE 6 SPACE ALLOCATION (T=TRKS, C=CYLS, B=BLKSIZE) The default means:- Supply Primary and Secondary space allocations, but do not supply DCB information as a default.
TITLE	*see notes*	*Alphanumeric string (see notes)*	The string specified is printed on the heading line of every Jol compiler listing. The default title is: ***** JOL *** DEMONSTRATION SYSTEM.**
PE	**NO**	**YES** *or* **NO**	**PE=YES** specifies that all Jol code produced by a MACRO command is to be printed in the second compiler listing with it's symbolic parameters replaced. Specify PE=NO if such a printout is not required. *This parameter need not be specified.* It can be overridden by JOL Compiler or Command Processor options
PI	**YES**	**YES** *or* **NO**	**PI=YES** specifies that the Jol code included with an INCLUDE command is to be printed as part of the compiler's output listing. Code **PE=NO** if the printout is not required. *This parameter need not be specified.* It can be overridden by JOL Compiler or Command Processor options.
PJCL	**NO**	**YES** *or* **NO**	**PJCL=YES** specifies that the generated control statements are to be printed with the jobs printed output. If the printout is not required code PJCL=NO. *This parameter need not be specified.* It can be overridden by JOL Compiler or Command Processor options.

Parameter	Default	Allowable Values	Notes
PM	**NO**	**YES** *or* **NO**	**PM=YES** specifies that Jol Macro Code will be listed on the compiler listing as it is being interpreted. That is, Jol Macros will be printed as they are being read from the Macro library. *This parameter need not be specified.* It can be overridden by JOL Compiler or Command Processor options.
PO	**NO**	**YES** *or* **NO**	The **PO** parameter specifies whether the final setting of all option switches, plus statistics on the number of accesses to macro files and the type of compiler the system is running should be printed on the compiler output listing. If required code PO=YES. *This parameter need not be specified.* It can be overridden by JOL Compiler or Command Processor options.
PRINT	**YES**	**YES** *or* **NO**	This parameter specifies whether or not to produce a listing from the Jol compiler. If the parameter is not specified, printout will be produced.
PRNTALL	**YES**	**YES** *or* **NO**	Specifying **PRNTALL=YES** results in printing of *all* JCL control statement allocation messages. Coding **PRNTALL=NO** will cause allocation messages to print **only** when an abend occurs. If nullified, i.e. **PRNTALL=**, is specified the installation reader parameter for MSGLEVEL will apply.
PRNTJCL	**NO**	**YES** *or* **NO**	**PRNTJCL=YES** specifies that the control statements are to be printed when the job executes. If nullified with **PRNTJCL=**, the installation reader parameter will apply.
PRTY	*null*	*any one or two digit number (in range 1-15)*	**PRTY** specifies the default PRTY value to be used on the job card when the user does not specify a priority. The null default means that the installation JCL reader default parameter will apply.
PUNCHCL	**B**	*a valid default* **SYSOUT** *class for punched output*	When a programmer specifies a PUNCH data set Jol converts this specification to **SYSOUT=***class*. This parameter MUST be specified.

Parameter	Default	Allowable Values	Notes
REALJCL			Not implemented on Jol 3.5.
REL	**3.8**	**OS release number**	This specifies the release number of the operating system e.g. REL=3.8.
SCHED	**$JOLSH60**	*alphanumeric names*	This specifies the Scheduler/Monitor name. It is principally for use by the developer, to test and develop Jol.
SM	**1,72,00**	*numeric values*	The **SM** parameter specifies the starting and ending columns for Jol language statements, and the optional column for print control characters. This parameter must not be nullified with **SM=,**.
SPCFORM	**S**	*any one character*	This parameter allows an installation to specify what special stationery unique to their needs is to be used. When a user declares a PRINT file, he may specify that the printed output is to go to a Special Form. For example, if the following statement is coded:- **DCL PRINTER PRINTER FORM 100;** the character coded for the SPCFORM parameter is placed in front of FORM coded (the 100 in this case), and thus the SYSOUT form becomes **S100**.
SPOOL	**JES2**	**JES2, JES3**	Specifies what subsystem will do the spooling. Note that the Symbolic Variable **%SPOOL** is set to the value coded. Various Jol Commands use the values of **%SPOOL** and **%SYSTEM** to determine which control statements should be generated, as some commands must generate different control statements depending on the Operating System.
SPOOL	**see notes**	**HASP, ASP, JES1, JES2, OR JES3**	Specifies what subsystem will do the spooling. Specify HASP or ASP only with MVT or MFT. MVS defaults to JES2. VS1 defaults to JES1. Note that the Symbolic Variable **%SPOOL** is set to the value coded. Various Jol Commands use the values of **%SPOOL** and **%SYSTEM** to determine which control statements should be generated, as some commands must generate

Parameter	Default	Allowable Values	Notes
			different control statements depending on the Operating System.
SRDR	**NO**	**YES** *or* **NO**	On some systems, Jol must start a System Reader to transfer the generated code to the System Job Queue. When **SRDR=YES** is specified, Jol will transfer control to a program which will enter Supervisor Mode and issue an SVC34 to START JOLRDR,DSN='dsname' where 'dsname' is the DSNAME of the Data Set containing the generated control statements. The following table shows the programs linked when **SYSRDR=YES** is specified. **SYSTEM TYPE PROGRAM EXECUTED** any, HASP=YES UJERDRH any, ASP=YES UJERDRA MVT UJERDRV MFT UJERDRF VS1 UJERDR1 MVS UJERDR2 For VS1, Jol usually uses a Jol supplied internal reader, similar to the reader supplied with MVS. This is not necessary on systems containing an internal reader (HASP, JES2 or JES3). This parameter need not be specified.
SYSBLK	**800**	*numeric value*	This specifies the default blocksize to be used if a blocksize is not specified by the Jol user. If the default parameter is altered the blocksize specified must be a multiple of 80 characters.
SYSCLAS	**A**	*any valid* **SYSOUT** *class for non-***TSO** *jobs*	This parameter specifies the default SYSOUT class for NON-TSO submitted jobs. (For TSO jobs see below). This parameter need not be specified.
SYSDEFS	*null*	*any valid sub-parameters*	The **SYSDEFS** sub-parameters are the defaults to be used for print files where the parameters are not coded by the user. Except under special circumstances, the installation operating environment options should be used and it is recommended that SYSDEFS be nullified initially. To nullify, code **SYSDEFS=,.**

Parameter	Default	Allowable Values	Notes
			The following are the sub-_parameters and their meanings. **Sub-parameter Meaning** 1 RECFM 2 BLKSIZE 3 LRECL 4 Primary Space 5 Secondary Space 6 Space allocation. (T=TRKS, C=CYLS, B=BLOCKS)
SYSTEM	**MVS**	**MFT, MVT, VS1, MVS**	This specifies the Operating System Jol will be running under at a particular installation. This parameter need not be specified. The Jol symbolic parameter **%SYSTEM** will contain the value coded, and may be accessed in Jol code to deteremine the Operating System being used.
SVC	**13**	**any SVC number (In range 0-255)**	The SVC parameter informs Jol of the number of the SVC to be executed when it wishes to enter the Supervisor Mode. On MVT and MVT systems enter Supervisor Mode only to START READER (see 38 SRDR above). VS1 uses Jol's own Internal Reader. Note: For MVS and VS1 Control Programs, MODESET is used rather than the Jol SVC. Any SVC-number between 0-255 is permissable. The SVC number may be 1-3 digits. This parameter need not be specified. _I
TSOCLAS	**X**	**any valid SYSOUT class**	This parameter specifies the default MSGCLASS for TSO submitted jobs. Note: All Jol commands default to **SYSOUT=*** for printed output. For VS1 systems, this is then changed to the MSGCLASS before submitting the job. Any printed output directed to **SYSOUT *** will be

Parameter	Default	Allowable Values	Notes
			returned to the TSO terminal unless a non-TSO MSGCLASS is specified. For jobs compiled in Batch, the printed output will also be returned to the MSGCLASS.
TYPE	**CSECT**	**CSECT or DSECT**	The JOLGEN macro may be used by other components of Jol and these, if used, require a DSECT to be generated. This is primarily for use by the supplier in testing and developing Jol. This parameter need not be specified.

Sample JCL and macro definitions are contained in the Loading Instructions and in the JOLINSTL data set provided.

3.3 The VOLSTART Macro

The **VOLSTART MACRO** is used to describe volumes and units. In order to generate efficient control statements for the Operating System Jol requires the following information:-

(a) Which volumes are permanently mounted.

(b) The unit types and the resultant unit name to be substituted when the user codes 'TAPE' or 'DISK' etc.

To assist the user when coding Data Set Declare Statements, Jol allows the use of "Preferred" or known volumes. This allows a user of Jol to code specific volumes without the VOL keyword. Jol recognizes this as a valid volume and automatically supplies the correct unit. In effect, the volume name becomes part of the Jol language.

To set up the table for Jol (the Volume Attribute Table) the VOLSTART macro is used, followed by the other macros described below, terminating with a VOLEND macro, in this manner:-

VOLSTART
    volume macro definitions
VOLEND

The **VOLSTART** and **VOLEND** macros ***must be coded***, even if an installation decides not to inform Jol about any of its volumes.

Jol can also mark all non-permanent volumes as **PRIVATE** so that at JOB termination all volumes that are still mounted by the system for the job are unloaded. However, facilities are included to ensure that certain (or all, if desired) volumes will remain mounted after their last use in a job unless the units are required by the Operating System for other volumes or another job. See the **RETVOLS** macro for details.

3.3.1 Defining Permanent Volumes

When Jol recognizes a Permanent Volume it:

(i) does not generate control statements to ensure the Volume is mounted when **SCRATCH**ing or **DELETE**ing a data set on that volume.

(ii) leaves the pack mounted for the operator after its last use.

The following macros allow definition of permanent volumes. They are coded as shown, followed by a list of volume names. For example:-

```
        P3350 (111111,222222)
```

defines volumes 111111 and 222222 to be permanently mounted 3350 volumes.

Coding the list of volume definitions also permits Jol to thoroughly validate the VOLUME references, and check that UNITS coded match the volume.

MACRO DEFINITION	MEANING
P3380	Permanently mounted 3380 Disk Units
P3375	Permanently mounted 3375 Disk Units
P3350	Permanently mounted 3350 Disk Units
P3330	Permanently mounted 3330 Disk Units
P2314	Permanently mounted 2314 Disk Units
P2311	Permanently mounted 2311 Disk Units
P2301	Permanently mounted 2301 Disk Units
P2302	Permanently mounted 2302 Disk Units
P2303	Permanently mounted 2303 Disk Units
P23051	Permanently mounted 23051 Disk Units
P23052	Permanently mounted 23052 Disk Units

Figure 3-2. Describing Permanent Volumes

For example:-

```
      VOLSTART
      P3350 (SYSRES,WORK01,WORK02,333333)
      VOLEND
```

3.3.2 Defining "Preferred" volumes.

Just as with the permanent volume macros above, the preferred or known volumes are coded followed by a list of volumes. As with permanent volumes, the user need only code the volume name without the VOL keyword and Jol will assign the correct device or unit type.

MACRO DEFINITION	MEANING
D3330	Non-permanently mounted 3330 Disk Units
D2314	Non-permanently mounted 2314 Disk Units

D2311	Non-permanently mounted 2311 Disk Units
D2301	Non-permanently mounted 2301 Disk Units
D2302	Non-permanently mounted 2302 Disk Units
D2303	Non-permanently mounted 2303 Disk Units
D23051	Non-permanently mounted 23051 Disk Units
D23052	Non-permanently mounted 23052 Disk Units
T24001	2400-1 Series tape volumes
T24002	2400-2 Series tape volumes
T24003	2400-3 Series tape volumes
T24004	2400-4 Series tape volumes
T34002	3400-2 Series tape volumes
T34003	3400-3 Series tape volumes
T34004	3400-4 Series tape volumes

Figure 3-3. Describing Preferred Volumes

For example:-

```
VOLSTART
D3350 (454545,DBPACK)
VOLEND
```

3.4 *C) The RETVOLS Macro*

The **RETVOLS** macro is used to define non-private volumes, for specific purposes.

With the exception of Volumes described as PERMANENT (see **VOLSTART** Macro above), Jol will normally inform the operating system that a volume is PRIVATE. Thus, the volumes will normally be dismounted either after their last use in the job or at job termination.

However, some volumes may be required across job boundaries, and these may be specified with the **RETVOLS** macro.

The **RETVOLS** macro is coded followed by a list of volumes to be retained on the system at job termination. For example:

```
RETVOLS (DISK01,DISK02)
```

It is possible to code **RETVOLS ALL** or **RETVOLS NONE**, meaning that either ALL the volumes will have PRIVATE or NONE of the volumes will have PRIVATE.

3.4.1 Notes

1. Tape volumes may *not* be retained in this manner.

2. Do not code any 3350 or 3380 type volume names as they cannot be dismounted.

3.5 The DEFUNIT Macro

The Jol compiler checks all unit names for validity by calling a System routine. It also has its own table to check such units as 2314, 3330 etc. However, users generally prefer to specify units as **TAPE**, **DISK** and **SYSDA**, and thus Jol must be informed what units are defined on a particular system. It may also be necessary to specify units such as Card Readers, Card Punches, Printers and specific Tape and Direct Access units to the Jol compiler.

This is done with the **DEFUNIT** Macro.

The **DEFUNIT** macro is in the form:-

DEFUNIT *subparameter=(definitions)*

The "*definition*" fields of the subparameter are in the form:

(*User-name,JOL-output-name*)

For example, if the name TAPE was to be assigned to 2400-3 unit type and DA1 (Direct Access Unit 1) to a permanently mounted 3330 device, the following would be coded:

DEFUNIT T24003=(TAPE,2400-3),P3330=(DA1,130)

The first definition (TAPE) is recognized by Jol when the programmer codes it in a Data Set Definition; Jol then alters it to the second name (2400-3) before outputting the control statements, unless this second name has, in turn, been altered by the **SWAPUNIT** macro (see E below). The second definition (DA1) is treated similarly.

In the above example, had the

(TAPE,2400-3)

been coded as

(TAPE,TAPE),

the name **TAPE** would have been used as the **UNIT** allocation instead of 2400-3.

Note: The following names, if not supplied, will be supplied as defaults; they should be specified to Jol as Generic Names. They are used in Jol supplied macros, and it is expected they will generate valid unit names.

TAPE
DISK
PUBLIC
SYSDA.

The following subparameters are allowed in the DEFUNIT macro:

MACRO PARAMETER	**MEANING**
P3380	Permanently mounted 3380 Disk Units
P3375	Permanently mounted 3375 Disk Units
P3350	Permanently mounted 3350 Disk Units
P3330	Permanently mounted 3330 Disk Units
P2314	Permanently mounted 2314 Disk Units
P2311	Permanently mounted 2311 Disk Units
P2301	Permanently mounted 2301 Disk Units
P2302	Permanently mounted 2302 Disk Units
P2303	Permanently mounted 2303 Disk Units
P23051	Permanently mounted 23051 Disk Units
P23052	Permanently mounted 23052 Disk Units
D3330	Non-permanently mounted 3330 Disk Units
D2314	Non-permanently mounted 2314 Disk Units
D2311	Non-permanently mounted 2311 Disk Units
D2301	Non-permanently mounted 2301 Disk Units
D2302	Non-permanently mounted 2302 Disk Units
D2303	Non-permanently mounted 2303 Disk Units
D23051	Non-permanently mounted 23051 Disk Units
D23052	Non-permanently mounted 23052 Disk Units
T24001	2400-1 Series tape volumes
T24002	2400-2 Series tape volumes
T24003	2400-3 Series tape volumes
T24004	2400-4 Series tape volumes
T34002	3400-2 Series tape volumes
T34003	3400-3 Series tape volumes
T34004	3400-4 Series tape volumes

In addition the following subparameters are allowed:

CR Card Reader
CP Card Punch
LP Line Printers

3.6 E) The SWAPUNIT Macro

3.6.1 SWAPUNIT

It is sometimes necessary or desirable to have Jol generate different UNIT names to that coded by a programmer or returned by the catalog lookup routines. It might be decided, for example, to bypass AVR on TAPES but to allow it to function as usual for DISKS. Jol can change specified unit names to other unit names with the SWAPUNIT macro. This facility is not available with JCL submitted jobs as the catalog lookup routines over-ride any unit coded on the JCL statements.

However, Jol already searches the Catalog hence unit names may be changed to one or more that bypass AVR, for example TAPE instead of 2400-3.

Another use of the SWAPUNIT macro is when changing from one device type to another, for example 2314 devices to 3330 devices. By coding the correct SWAPUNIT operands, Jol will then generate the correct job control statements for the new system.

The SWAPUNIT macro is coded:

SWAPUNIT (*given-name,required-name*)

For example:

SWAPUNIT (2400-3,TAPE),(SYSDA,2314)

3.7 F) The NOCAT Macro

3.7.1 NOCAT

The **NOCAT** macro allows specifications of:

(a) data sets which are **<u>not</u>** to be located by searching the system catalog, either by data set name or high level indexes.

(b) data sets which are to be connected to Initiator Dedicated Data Set names. (See Initiator Dedicated Data Sets below.)

The **NOCAT** Macro is coded:

```
               GENERIC=generic-name
NOCAT          INDEX=high level index
               DSN=data set name
```

3.7.2 For example:

To stop Jol locating any data set starting with **SYS** or **GIS** code:

```
NOCAT INDEX=(SYS,GIS)
```

To stop Jol locating any data sets starting with **SYS1.** code:

```
NOCAT INDEX=SYS1
```

To stop Jol locating any data set **SYS1.LINKLIB** *and* **TSO.CMDPROCS** code:

```
NOCAT DSN=(SYS1.LINKLIB,TSO.CMDPROCS)
```

To stop Jol locating **SYS2.PROCLIB** and all data sets commencing with **SYS1** code:-

```
NOCAT DSN=(SYS2.PROCLIB),INDEX=SYS1
```

3.8 Initiator Dedicated Data Sets

An installation using MFT, MVT or VS1 may direct Jol to use Initiator Dedicated Data Sets. Normally, a user wishing to use Dedicated Data Sets must alter the Data Set Names that he codes in his JCL statements.

Jol can change the name of any data set coded to another. Thus an installation can, on a global basis, change data set names and force the use of Dedicated Data Set names. The programmer need not be aware that the names have been changed and the system should run more efficiently because Initiator Dedicated Data Sets will be used automatically. The time saved is often considerable because it takes time to allocate data sets, and the use of dedicated data sets simply forces the use of an already existing allocation.

4 *Generation of the Jol Command Processor*

4.1 The Command Processor

The Jol Command Processor allows users of TSO to easily invoke Jol interactively. It allocates user data sets and work files required by Jol, and calls the Jol Compiler with parameters that may be optionally specified when the Jol Command is entered at the terminal.

In order to satisfy individual installation requirements, the Command Processor is supplied in Source Code form, as an Assembler Macro. The macro is then assembled with various options (described below) to generate a tailored Jol Command for use under TSO or ISPF.

The new command Jol is then link-edited into a system library, thus making it available to all users. While it can be stored in the System Library as any name other than Jol, it is advised that the new command be called Jol to provide consistancy with the Jol documentation. An alias may be assigned, if desired.

Two, or more, Jol command processors may be set up (using different names) if it is desired to allow some users access to various versions and/or data sets. For example, Jol could be set up for the Programming Department, and PRODJOL set up (using other Jol data sets) for the Operations Department to submit Production work.

4.1.1 Pre-Allocated Data Sets

If any of the standard Jol ddnames are **already allocated,** $$JMACRO for example, to a library, the Command Processor will allow that allocation to be used. It will not free and re-allocate the data set. This permits an installation to use a CLIST to pre-allocate some of the Jol libraries and then invoke the Jol Command Processor to allocate the other data sets required by Jol.

All Jol compiler options and data sets may be specified at assembly time and most may be overridden at execution time. Data set names should be specified to the macro instruction following normal TSO conventions: if enclosed in quotes the name is understood to be fully qualified. If it is not enclosed in quotes, the name is prefixed with the TSO user's prefix as specified with the PROFILE command.

Additional facilities include:-

a) Allocation of the Jol compiler print file to a SYSOUT or DASD dataset.

b) Specification of the unit name to be used for the Jol compiler work data sets.

c) Specification of permanent and temporary data sets for the general control statements.

The Command Processor allows one or many data sets to be concatenated to the **INCLIB** or **CMDLIB** input files to Jol. An installation may specify as many as required.

Often, TSO users have their own Jol data sets. Each user **must** allocate a Jol data set before Jol may be used. Allocate **&SYSUID.JOL**, unless the installation decides to use a single Jol data set for all users, in which case the name must be specified in quotes so that Jol will not attempt to prefix the names by **SYSPREF** or **SYSUID**.

4.2 Assembly of the Command Processor

Assembly requires access to **SYS1.MACLIB** and **SYS1.AMODGEN**. The load module should be link-edited with the linkage-editor options **RENT, REFR, REUS.**

A sample job to assemble the Command Processor is provided in the **INSTALL** data set, and is described in the Loading Instructions for Jol.

4.2.1 Assembler Macro Syntax

The syntax of the assembler macro is:

name **JOLTSF** *option-list*

where:-

name is in the label field and is required. The first three characters are used to generate the Jol Command Processor message numbers as well as the names for several other control sections.

JOLTSF is the macro name and must be coded as shown.

option-list may contain the following keyword parameters. The options are described in table 5.

Note that all Jol compiler options and data sets may be specified using the parameters described in the option list, and that most may be overridden at execution time. Data set names should be specified to the macro instruction following normal TSO conventions: if enclosed in quotes the current user prefix is not used when allocating data sets.

Parameter	Default	Allowable Values	Notes
ALIAS	**JOL,J**	*any valid program name(s)*	**ALIAS** specifies the alternative names for the generated command processor. These are established with the linkage editor '**ALIAS**' statement, at assembly. If the command processor is subsequently re-link-edited you will need to supply your own alias statement. One, or a list of alias names may be specified. For example **ALIAS=(JOL,J)** If this parameter is not required it should be nullified i.e. specified as **ALIAS=,**.
CAT	**YES**	**YES** *or* **NO**	Specifies whether or not Jol is to search the catalog at compile time. **NO** specifies that catalog searching is to be performed at execution or run time. A partial search of catalogs is not permitted. **CAT** *must* be specified, and cannot be nullified.
C1,C2,C3	*null*	*any valid* **DD** *statement*	It is possible to specify up to three (3) DD cards. These cards are then generated each time Jol outputs a new OS job step. This facility allocates the specified data sets to every Jol step. The data sets can then, for example, be updated to show the current step that has just executed, thereby making information available for automatic restart. The format is: **C**$_n$='*ddname* **DD DSN**=*name*, etc' Example: **C1='RUNFILE DD DSN=SYS1.RUNFILE, DISP=SHR'**
CMDLIB	***userid*.JOL, SYS2. JOL60. CMDLIB**	*Any Macro data set names*	**CMDLIB** specifies the default macro data set or data sets to be associated with the Jol command processors MACLIB keyword. The data sets specified here are used whenever Jol requires a Macro Command. Example: **CMDLIB=(JOL,'SYS2.STND.CMDLIB', 'SYS2.JOL35.CMDLIB').** This need not be specified. If specified it *must not* be nullified. It may be overridden at execution time, unless

Parameter	Default	Allowable Values	Notes
			the data set has been pre-allocated.
DC	**null**	*any valid* **DD** *statement*	Specifies the default dump card to be used by Jol. It *may not* be a list. It may be changed with the **ON ERROR** Jol command or with the **DC** option on the Jol command. **DC** need not be specified.
DYNAMIC	**NO**	**YES** or **NO**	Specifies that Jol is not to generate real JCL, but to create a special instruction file for the Dynamic Allocation Jol Scheduler so that jobs are generated without JCL. In fact, a small amount of JCL is generated to start the jobs, and to enque on data sets. **DYNAMIC** is usually specified at execution time.
FLAG	**0**	**0, 1-73**	This parameter permits an installation to specify that an asterisk be printed at the column following the last column of the source code. This assists programmers to find errors due to coding outside the source margins. This parameter is usually omitted.
GENCB	**NO**	**YES** *or* **NO**	This specifies whether the OS mapping macros are to be expanded in the Assembler listing. This parameter *must not* be nullified i.e. do not specify **GENCB=,**.
INCLIB	***userid*.JOL**	*any valid data set name(s)*	**INCLIB** specifies the name or names of the default library or libraries to be associated with the command processor **INCLUDE** keyword statement. The **INCLIB** parameter *must not* be nullified i.e. do not specify **INCLIB=,**. It may be overridden at execution time, unless the data set has been pre-allocated.
JCL	*temporary data set*	*any sequential data set name*	This parameter specifies the default data set name for the Jol-generated control statements. It may be overridden at execution time *unless* a null value

Parameter	Default	Allowable Values	Notes
			(ie. **JCL=**,) is specified. If a null value is specified, the generated control statements are written to a temporary data set and, for security reasons, cannot be processed by the EDIT command processor or kept for re-use. A sequential data set must be specified, *not* a member of a PDS.
JCLBLK	**6080**	*numeric block size*	This specifies a default block size to be used for the data set containing the generated control statements if a block size is not specified by the user. If this parameter is altered the blocksize specified must be a multiple of 80 characters.
JLOAD	*null*	*load module library name(s)*	**JLOAD** specifies the name or names of the Jol load libraries. A single library or a list of names may be specified. If a null default i.e. **JLOAD=**, is specified the **SYS1.LPALIB** or **SYS1**.*installation* libraries must already hold a copy of the Jol compiler. It may be overridden at execution time, unless the data set has been pre-allocated.
LET	**YES**	**YES** or **NO** or *Numeric Value* or (**YES** or **NO** or *Numeric Value*, **PROMPT** *or* **NOPROMPT)**	**LET** specifies if the **LET** compiler option is to be selected by default. If a numeric value is specified, and the compiler returns a return code greater than that specified the user will be queried as to whether the job should be submitted. **LET=YES** specifies continue regardless of errors occurring. **LET=NO** specifies **NLET** i.e. exit if errors are detected. **PROMPT** specifies that the User is to be asked if the job should be submitted if errors occur, **NOPROMPT** specifies that the User will not be asked if the job is to be submitted if errors occur.
LIST	*null*	*any valid* **SYSOUT** *class*	**LIST** specifies the default SYSOUT class for Jol compiler messages, for when the user selects the

Parameter	Default	Allowable Values	Notes
			LIST parameter on the Jol commands entered from the terminal. It must be a single, valid alphanumeric character in use at the installation. It may be overridden at execution time. Normally, Jol listings are output to the data set specified by the PRINT parameter.
OS	**MVS/XA**	**MVS/XA** or **MVS**	OS specifies the version of OS for which the command processor is being generated. It is necessary to direct the assembly to select the correct mapping macros.
PCOM	**YES**	**YES** *or* **NO**	This parameter specifies whether comments are to be printed in the generated JCL. This parameter need not be specified. It may be overridden at execution time.
PEXPAND	**NO**	**YES** *or* **NO**	Specifies whether the **PEXPAND** option should be selected by default. This parameter need not be specified. It may be overridden at execution time.
PGM	**$JOLCP60**	**JOL Compiler name**	**PGM** specifies the Jol compiler name. It ***must not*** be the name of the Jol command processor as stored in the system library. It must be correctly specified; an incorrect specification may cause an 806 abend or other undesirable results. This parameter should not be altered or overridden, unless testing a new version of the Jol compiler.
PINCLUDE	**YES**	**YES** *or* **NO**	Specifies whether to print the 'included' Jol. This parameter need not be specified. It may be overridden at execution time.
PJCL	**NO**	**YES** *or* **NO**	This specifies whether job control statements

Parameter	Default	Allowable Values	Notes
			are to be printed on the Jol listing file at compile time. This parameter need not be specified. It may be overridden at execution time.
PMACRO	**NO**	**YES** *or* **NO**	**PMACRO** specifies whether the print macro option should be selected. **PMACRO=YES** means force the Jol compiler option PM. **PMACRO=NO** means do not force the PM option. This parameter need not be specified. It may be overridden at execution time.
POPT	**NO**	**YES** *or* **NO**	This parameter specifies whether Jol options are to be printed. This parameter need not be specified. It may be overridden at execution time.
PREFOUT	**USERID**	**USERID** *or* **SYSPREF**	**PREFOUT** specifies the preferred prefix for the data set containing the job control statements and the compiler print file output file. This parameter need not be specified.
PRINT	*userid.* **JOL.LIST**	*data set name* *or* * *or* **'NULLFILE'**	**PRINT** specifies the default print data set for the Jol compiler messages. If * is specified the print output will be directed to the terminal. If '**NULLFILE**' is specified then no print or listing will be produced. A data set name should be specified ***without quotes*** to allow normal TSO defaulting to apply. Note that a print data set will always be on a data set, unless the **LIST** option is selected at execution time to direct the file to SYSOUT data set, or the * option is specified, thus directing the listing to the screen.

Parameter	Default	Allowable Values	Notes
PRTBLK	**6144**	*numeric block size*	Specifies the default print file block size, to be used where blocksize is not specified by the user. If the default parameter is altered the blocksize specified must be at least 125 characters. The file is VBA.
SUB	**NO**	**YES** *or* **NO**	**SUB** specifies whether or not the **LET** compiler option is to be specified by default. **SUB=YES** forces the Jol option LET and submits the generated code regardless of the Jol compiler's return code value, even though errors may have been detected. **SUB=NO** allows the Jol option LET to be specified separately. That is, if the Jol return code is zero submit the generated statements. If the return code is non-zero test the LET specification. If yes, process. If no, exit if Jol errors are detected.
SYMS	**TSOCLASS =X**	*any character string*	Specifies a set of symbols to be passed to the Jol compiler. These variables may be tested or used in any Jol code. For example; **SYMS='TSOCLASS=X'** will cause the variable **%TSOCLASS** to have an initial value of X. Note that the character string must be enclosed in quotes. **SYMS** may be overridden at execution time. Note that the Command Processor always sets the Symbolic Variables **%SYSPREF** and **%SYSUID** to the appropriate values.
SYNCHK	**NO**	**YES** *or* **NO**	This specifies if syntax checking is to occur. Specifying **YES** instructs the Command Processor *not* to submit the job, that is to perform only a syntax check. As this parameter can be specified when the Jol Command is invoked, it is advised

Parameter	Default	Allowable Values	Notes
			that the default be left as **NO**, unless another Command Processor, say **JOLTEST**, is set up with **SYNCHK=YES**.
TERM	**YES**	**YES** *or* **NO**	Specifies whether the Jol compiler option **TERMINAL** is to be selected. Note that the Jol compiler selects this option based on whether the file SYSTERM is allocated. The command processor does not free the file SYSTERM if it is allocated, so TERMINAL may be selected even if **TERM=NO** (or NOTERMINAL at execution time) is selected.
TEST	**YES**	**YES** *or* **NO**	**TEST** specifies whether the TEST option should be selected by default. The TEST option results in the Jol compiler asking whether to submit the Jol-generated code. **TEST=YES** selects the Jol compiler option TEST. **TEST=NO** specifies the Jol compiler option NTEST. The TEST specification may be overridden at execution time.
TSO	**NO**	**YES** or **NO**	Specifies if the job stream is to be executed *immediately under TSO* or submitted to background to execute. For Fujitsu systems running PFD without the EDIT macro facility, you can generate a Jol command processor and call it **RUN** - thus when the User types **RUN** when editing a Jol data set, Jol is called and actually runs the job immediately under TSS. For MVS Users, it is recommended that an EDIT macro be wriiten to call the Jol command processor with the TSO option on the command line. **TSO** is usually specified at execution time.
UNITJCL	**VIO**	*any valid unit name*	**UNITJCL** specifies the unit name to be used by the Jol compiler for the allocation of the job control statements file (if it is a temporary data set). If the installation uses VIO it is recommended, unless extremely large Jol jobs are being

Parameter	Default	Allowable Values	Notes
			submitted. If a null value (**UNITJCL=,**) is supplied eight blanks are assigned to allow the installation default options to be applied.
UNITWRK	**VIO**	*any valid unit name*	**UNITWRK** specifies the unit name to be used for temporary data set allocation by the Jol compiler work data sets. If the installation uses VIO it is recommended. If a null value (**UNITWRK=,**) is supplied eight blanks are assigned to allow the installation default options to be applied.

Notes

1. There is one user exit point provided in the Jol command processor. It is optional, and if provided, must be link edited into the same load module as the Jol command processor. It must be re-entrant and reuseable.

   It is passed one parameter, in accordance with the standard OS conventions. This parameter is an area into which the exit may place up to 256 bytes if symbols are to be passed to the Jol compiler.

   The following information is required upon return:

a) A return code in register 15.

0 means call the Jol compiler, no additional symbols are provided.

4 means call the Jol compiler, additional symbols are provided.

8 means do not call the Jol compiler.

b) Symbols acceptable to Jol (return code 4 only). These symbols are in the area provided to the exit. The first two bytes contain a binary number indicating the number of bytes in the string of symbols. This character string is joined to the parameters already predefined to pass to the Jol compiler.

A sample user exit is provided with the Jol command processor. It was written for a site where the accounting information for **TSO** users consists of eight bytes; the first four are the charge code and the last four are the room number. The information is located in the Job Account Control Table and is returned as two symbols, **SYSACCT** and **ROOM**.

2. In order to use the Jol command processor, the user must be authorized to use the SUBMIT command. Refer to the TSO ACCOUNT command for information on how to achieve this.

3. Jobs are submitted by passing control to the standard SUBMIT command processor.

4. The following user information is provided to the Jol compiler as system variables.

   SYSUID and **SYSPREF**

   These are taken from the TSO values. The sample user exit also provides

SYSACCT and ROOM

These are taken from account values stored in the user attributes data set or provided at LOGON.

5 Post Installation

5.1 Testing Jol

After installation of Jol, tests should be carried out to ensure that the compiler and execution modules are functioning correctly. You can do interactive and batch processing tests.

5.1.1 INTERACTIVE PROCESSING

To test Jol in an interactive mode, you must first allocate a Jol data set. You can use SPF to do this with menu 3.2. The block size of the data set must be the same or larger than **SYS2.JOL60.INCLUDE**.

If you the **TSO COPY** comand, you can use it to create your own Jol data set as in the following example:-

```
COPY 'SYS2.JOL60.INCLUDE' JOL
```

then enter

```
JOL *
```

You will receive the message:

'Please enter your Jol statements and end with a /*'

then enter:

HELP; (Jol will display HELP information)

or **CAIJOL**; (the Jol Teach Yourself Jol will be invoked)

or **BUILDJOB;** (the BUILDJOB processor will be invoked).

5.1.2 BATCH PROCESSING

To test Jol running as a batch job, use the JCL and Jol code shown in figure 6 to list the contents of SYS2.JOL60.CMDLIB and the validation exits provided with Jol.

```
//JOB etc.
//EXEC JOLCGO
  LIST:JOB 60 K 2,1 MINS;
  PRINTPDS SYS2.JOL60.CMDLIB;
  PRINT    SYS2.JOL60.SOURCE(UJUVALID);
  PRINT    SYS2.JOL60.MACLIB(UJUEXIT) ;
  PRINT    SYS2.JOL60.MACLIB(USERCOMS);
```

6 *Writing Invoked Routines*

Adding New Instructions to Jol

An **INVOKE**d routine is one written in COBOL, PL/1 or Assembler that conforms to special Jol conventions. An **INVOKE**d routine effectively becomes part of the Jol *compile time* language just as a Macro Command becomes part of the Language

Reasons for using INVOKEd routines.

Even though Jol has inbuilt intructions to **ALLOCATE, READ** and **WRITE** data sets, and the Macro Command Facility of Jol is extremely powerful and easy to use, there are certain functions that can only be done by invoking a program or subroutine.r
For example, a high level language or assembler routine must be used to search a data base and set up a Symbolic Variable with the contents of a record from that data base, or query some part of the Operating System (such as the catalog).

An **INVOKE**d Routine may:-

1. Generate any Jol instruction or Macro Instruction including another **INVOKE** - (explicitly or implicitly). Jol will then execute the instructions as though they had been read from one of the input streams.
2. Act as a Jol Instruction itself and alter various Symbolic Variables and output instructions to the Compiler Phase through the use of Jol subroutines.
3. Combinations of the above.

INVOKEd Routines gain control either:-

1. Explicitly, through the use of the Jol **INVOKE** or **CALL** instructions.
2. Implicitly, when a non-Jol instruction is found, and that instruction is not in the Macro Library, but a Load Module of that name is found in the data sets referred to by a DDCARD $$JLOAD.

   NOTE: If the subroutine uses any data sets, they must be allocated before they can be used. The Jol or TSO **ALLOCATE** instructions can be used, or a JCL DDcard added to any JCL used to call Jol to enable the routines to gain access to the data sets.

Returning Information to Jol

To return statements to Jol from an **INVOKE**d routine, call the **UJPLIOP** subroutine. This routine may be called from PL/I, COBOL or FORTRAN and it returns to Jol the characters in the string. Assembler routines may call the **UJP98OP** routine, with increased efficiency.

When your routine terminates, Jol interprets the data returned in the strings as if they had been read from one of the input streams.

PL/I routines call **UJPLIOP** with a varying length string as a parameter. The string may be up to 32,000 characters in length.

Other languages must simulate the varying length string before calling **UJPLIOP**. This is simply done by preceeding the characters string with a two byte (halfword) binary number, and setting it to the length of the string before calling **UJPLIOP**.

Notes on how Returned Instruction are Handled

INVOKEd routines can:

- Alter Internal Variables in the Jol Compiler.
- Send instructions back to Jol that are then executed as normal Jol instructions.
- or a combination of the above.

When returning instructions to Jol, the following rules apply:

1. Jol interprets the instructions as though it had read them from a Jol Macro. The instructions are treated exactly as a normal Jol Macro instruction, including any Print Options specified to the Compiler.

   Therefore, when returning any instructions to Jol through the **UJPLIOP** or **UJP980P** subroutines, the first instruction *must be* a **MACRO** instruction, without parameter prototypes specified. You may follow this immediately with an **END** statement if your instructions are designed to be processed at the same *level* as the code that called your routine.

   This is the only method by which the *level* can be raised. Therefore, any symbolic variables that are declared become *local* variables and will be deleted when your routine terminates. Similarly, any program or data sets declarations will also be removed from the Jol symbol table at the termination of the Routine.

2. Any returned instructions are written to the Work-file for later interpretation. They *are not interpreted* as soon as they are passed to Jol.

3. If your routine returns a non-zero value in **R15**, Jol will ignore any instructions returned via the transfer routines.

6.1 Writing a PL/1 Invoke Routine

The first PL/1 statement must be a **PROC OPTIONS(MAIN)** statement. Any other PL/1 statement may follow.

Example 1 of a PL/I Invoke Routine

For example, suppose that a new instruction called **ASKOP** is required. Its function is to ask the operator a question, and return the answer to the Jol procedure that called it. Such a routine could be coded as follows:-

```
ASKOP: PROC(PARM)          OPTIONS(MAIN);
          DCL PARM         CHAR(100)VAR;
          DCL ANS          CHAR(100)VAR;
          DCL UJPLIOP      ENTRY (CHAR(*)VAR);

          CALL UJPLIOP('MACRO;');            /* START MACRO */
          CALL UJPLIOP('END;');        /* END MACRO */

          DISPLAY(PARM) REPLY(ANS);          /* ASK OPERATOR */
          CALL UJPLIOP('SET ANS='''||ANS||''';');        /* RETURN ANSWER */

END;                 /* END OF PROGRAM */
```

Figure 6-1. Example 1 of a PL/I INVOKE Routine

To use the routine to ask the operator a question, code the following in Jol:-

```
          ASKOP 'DO YOU WISH TO RUN SALES TODAY';

          IF %ANS='YES'
                   _THEN INCLUDE SALES;

          ELSE SIGNAL 1,'SALES NOT TO BE RUN TODAY';
```

The example above shows how to use the UJPLIOP routine to return information to Jol. PL/1 routines can be any size (subject to storage restraints), and can access OS data sets.

Example 2 of a PL/I Invoke Routine

A more elaborate example follows. In this case, the routine must read a record from a data set, and perform a simple validity test. If the record is acceptable, then the value in the record is to be compared with the parameter and **EQUAL** or **UNEQUAL** is to be in a symbolic variable called **%CONTROL**. Should the validity check fail, a **STOP** command is to be returned to Jol - this being **STOP 'CONTROL RECORD IN ERROR - GREATER THAN 5'**, followed by the record read. Note that the **GET** instruction could be used for this particular function.

```
CHKFLE: PROC(PARM) OPTIONS(MAIN);
            DCL UJPLIOP     ENTRY (CHAR(*)VAR);
       DCL PARM             CHAR(100)VAR;          /* PARM FROM
                              INSTRUCTION */
       DCL INFILE     FILE RECORD INPUT;           /* DEFINE FILE */

       DCL PARMRECORD
            CHAR(80) STATIC;       /* I/O AREA    */

            CALL UJPLIOP('MACRO;');
            CALL UJPLIOP('END;');

            READ FILE(INFILE) INTO(PARMRECORD);
            IF PARMRECORD > 5              /* RETURN ERROR */
            THEN DO;
                 CALL UJPLIOP('STOP '''||
                     'CONTROL RECORD IN ERROR - GREATER THAN 5'
                     ||PARMRECORD);
            END;
            ELSE DO;
                 IF PARM=PARMRECORD
                 THEN CALL UJPLIOP('SET CONTROL=''EQUAL'';');
                 ELSE CALL UJPLIOP('SET CONTROL=''UNEQUAL'';');
            END;
        END;
```

Figure 6-2. Example 2 of a PL/I INVOKE Routine

To use **CHKFLE** routine, Jol code similar to the following could be used:-

```
       INVOKE CHKFLE 1;
       IF %CONTROL='EQUAL' THEN .....

  or more simply:-

       CHKFLE 1;
       IF etc
```

6.2 Writing a COBOL Invoke Routine

COBOL may also be used to return information to Jol. **COBOL** does not have variable character strings as such, but these can be simulated by setting the length of the string to be returned as a binary halfword (eg PIC S9(4)) and setting the length of the string before calling **UJPLIOP**. An example of using COBOL to return a string **%A='OFABAN.CNTL(TESTJOL)'** follows.

```
      ISN   SEQND   A      B

     1    000010  IDENTIFICATION DIVISION.
     2    000020  PROGRAM-ID PHILPROG
     3    000030  ENVIRONMENT DIVISION.
     4    000040     DATA DIVISION.
     5    000050     WORKING-STORAGE SECTION.
     6    000060    77  FILLER PIC X(24) VALUE  '  WORK FOR PHILPROG'.
     7    000070     01  DATA-AREA.
     8    000080         05  DA-COUNT PIC S9(4) COMP VALUE +1.
     9    000090         05  DA-CHAR.
    10    000100            10 DA-C PIC X OCCURS 0 TO 100 DEPENDING ON DA-
  COUNT.
    11    000110    LINKAGE SECTION.
    12    000120    01  PARM-L.
    13    000130        05  PARM-L PIC S9(4) COMP.
    14    000140        05  PARM-F PIC X(100).
    15    000150    PROCEDURE DIVISION USING PARM-F1.
    16    000160    START-AND-END.
    17    000170        MOVE PARM-L TO DA-COUNT.
    18    000180        MOVE PARM-F TO DA-CHAR.
    19    000181        MOVE 8 TO DA-COUNT.
    20    000182        MOVE 'X:MACRO;' TO DA-CHAR.
    21    000183        CALL 'UJPLIOP' USING DATA-AREA.
    22    000190        MOVE 27 TO DA-COUNT.
    23    000200        MOVE ' A=''OFABIN.CNTL(TESTJOL)'';' TO DA-CHAR.
    24    000210        CALL 'UJPLIOP' USING DATA-AREA.
    25    000211        MOVE 4 TO DA-COUNT.
    26    000212        MOVE 'END;' TO DA-CHAR.
    27    000213        CALL 'UJPLIOP' USING DATA-AREA.
    28    000220        GOBACK.
```

Figure 6-3. Example of a COBOL INVOKE Routine

To **INVOKE** the COBOL routine, simply code the name of the module. It will be loaded, and control transferred to it. If it was link-edited as INVCOB, the coding:-

INVCOB;

will execute the subroutine.

6.3 Writing an Assembler Invoke Routine

You can write your routines in Assembler instead of using PL/1 or COBOL. When using Assembler it is possible to directly access and alter many variables used internally by Jol.

As with PL/1 and COBOL, Jol instructions may be returned to Jol through a special routine. In addition, it is possible to utilize the Jol internal routines that define and/or alter symbolic variables. Using these functions will become clearer from a review of the examples.

6.3.1 Register Contents on Entry to the Routine

The following table shows the values of registers when your routine gains control:

R0	Undefined
R1	Address of a normal OS parameter list (see below)
R2	Contains the address of **JOLCOM**, the Jol Common Communications Table.
R3	Contains the address of **TKNX**, the area in which the current instruction has been Tokenised.
R4-12	Undefined
R13	Address of a dynamically linked save area.
R14	Return Address
R15	Entry Address (address of this routine).

Figure 6-4. **Register Contents on Entry to an INVOKEd Routine**

6.3.2 Special Notes on Register 1 Contents

When your routine is executed, Register 1 is set up to *appear* that the program was loaded by a JCL **EXEC** statement, or a **CALL** under Jol or TSO. This is done by setting the high order bit of the parameter **ON**, just as JCL would do for you, thus the program will assume that it was called via a JCL **EXEC PGM**, or via a **CALL** in **TSO**.

Notes

1. Register 1 points to a parameter area. The first word of the parameter area contains an X '80', followed by the address of the *first* parameter character string. If any normal OS program is **INVOKE**d, it assumes that it was loaded by the equivalent of a **RUN** in Jol, or an **EXEC** in JCL, because of the high order bit setting.

   The address portion of the first word points to a half-word containing the length of the Parameter, followed by the actual parameter. This is the same as normal OS conventions.

Examples of parameter contents:

a) **INVOKE** *module* **123**; the parameter contains:

H'3',C'123'

b) **INVOKE** *module* **'ABCD';** the parameter contains:

H'4',C'ABCD'

c) *module* **123**; the parameter contains:

H'3',C'123'

which is the same as a) above. Note that if the word **INVOKE** is not coded, Jol attempts to find a module of that name and load it, in exactly the same way as does had **INVOKE** been coded.

2. If **INVOKE MODX A,B,C;** is coded *without quotes*, Register 1 points to the **'A'** only. **'B,C'** are ignored but can be accessed through the use of the **GETTKN** Macro instruction.

3. Only programs or load modules written specifically for the **INVOKE** instruction may use the extra facilities.

4. All normal Jol and OS Macros may be used.

   To make the use of *any* of the Jol Macros or Subroutines, R2 must point to **JOLCOM**, and if the **GETTKN** Macro is used, R3 must point to **TKNX**. These are the default entry conditions.

5. Re-enterability has the usual advantages. If you wish to **LINK** your frequently used modules *permanently* into Jol to save loading time, they must be re-entrant.

6. Jol assembler macros are described in the Appendix of this manual, and also in the **USERCOMS** member of the Jol supplied **MACLIB**. These descriptions should be read for further details of the Macros.

7. To use the **JOLERR** Macro, your CSECT must be of the form:

   xxCNNxx where *CNN* is a unique name.

   The **JOLERR** macro uses the *CNN* name to produce unique message numbers. For example, if your CSECT name is **UJZ01CLR**, and it issues a **JOLERR 109, 'MODULE INVOKED'**, the Compiler will issue the following message:

   UJZ01-09 MODULE INVOKED

   The message number **UJZ01-09** is made up of the following components:-

UJ A constant. All Jol Compiler messages commence with **UJ.**

Z01 Taken from the ***CNN*** number described above.

- A constant.

09 A unique message identifier coded on the **JOLERR** macro.

See the **JOLERR** description for further details.

11. Module names from UJX01 to UJZ99 are valid for user-written invoked routines. You can use an **Alias** to specify meaningful command names for the Users of your special Jol routines.

    Other names are reserved.

12. To Declare or Assign a value to a Symbolic Variable, move the name of the Variable to the field called **SYMBOLIC** in **JOLCOM**, and the contents to **WORK.** In addition, set the **#WORK** half-word field to the length of the contents, and call the **UJP02DCL** or **UJP85ASN** subroutine.

    For example, to store **10** into Symbolic Variable **A**, you could code:

```
MVC SYMBOLIC,=CL8'A'
MVC WORK,=C'10'
MVC #WORK,=H'2'
$CALL UJP85ASN
```

    Note that **#WORK** is set to the *actual* length of **WORK**, not the length of the contents + 2. This is same method used in PL/1 varying strings.

13. The subroutine **UJP980P** returns an instruction to Jol. For example, to return a **STOP 'INVALID NAME';** instruction to Jol either:-

a)

```
        $CALL UJP980P,AREA
        ...
AREA     DC  AL2(L'STOPMS)
 STOPMS  DC C'STOP "INVALID NAME";'
```

or

b)

```
MVC WORK(20),=C'STOP "INVALID NAME";'
MVC #WORK,=H'20'
$CALL UJP980P,#WORK
```

Notes

i) The length is the *actual* length of the instruction to be transferred to Jol.
ii) Multiple instructions may be transferred.
iii) Each instruction must be terminated by a semi-colon.

iv) Instructions greater than 72 characters must be transferred in successive calls to **UJP980P**. This is an implementation restriction.

Deciding on a Method to Perform the Function

There are two methods you can use to transfer data to Jol.

Method 1: You call **UJP98OP** to transfer any Jol instruction back to Jol.

Method 2: You directly transfer Symbolic variable data back to Jol by calling **UJP85ASN** or **UJP02DCL**.

This method restricts you to transferring *symbolic data only*.

You must decide which method is to be used to transfer information to Jol.

Generally speaking, Method 1, transferring instructions to Jol is via **UJP98OP** is:-

1. Very flexible.
2. Easy.
3. Not quite as efficient as using the direct subroutines.

The type of functions to be performed also have a considerable bearing on the decision.

If the instruction is to be used on a frequent basis, it may be desirable to cut down on the amount of CPU time required to perform it by using a subroutine; but if on a non-frequent basis, the method of transferring instructions is easier and quicker to debug, as well as being easier to alter if changes are required.

6.4 Further Examples of INVOKEd Routines

Example 1: Writing an instuction to **CLEAR** a list of Symbolic Variables. The Syntax of the instruction is:-

CLEAR name-list ;

```
         COPY      JOLCOM
         USING     JOLCOM,R2
         USING     TKNX,R3
JOLSAVE  CSECT=UJX01CLR
*   TELL JOL THAT A MACRO PROGRAM IS IN CONTROL.  THIS
*   TELLS THE ASSIGNMENT ROUTINES NOT TO PARSE A STRING
*   AND TRY TO PERFORM ARITHMETIC WHEN UJP85ASN IS CALLED.
         MVC       ICOMMAND,=CL8'MACRO'
* IF THE NAMES OF THE LIST ARE NOT ALREADY DEFINED
* BY A 'DCL' WE SHALL IMPLICITLY DECLARE THEM
         GETTKN    2             GET FIRST NAME
         CLI       TKN,C' '      BLANK? IE NO NAMES IN LIST?
         BNE       X01LOOP       NO, START PROCESSING NAMES
         JOLERR    001,'NO NAMES IN CLEAR COMMAND' ERROR
         JOLRETN
X01LOOP  STH       R1,TKNCURR    SAVE TOKEN NUMBER BEING PROCESSED
         CLI       TKN,C' '      END OF LIST?
         BE        X01END        YES, END OF ROUTINE
         CLI       TKN,C','      ',' IS NAME LIST SEPARATOR
         BE        X01GETNX      YES, SKIP IT, GET NEXT ITEM
         CLI       #TKN+1,8      NAME TOO LONG ?
         BH        X01ERR2       YES, ERROR AND IGNORE IT
         CLI       TKNTYPE,2     IDENTIFIER (PROPER NAME?)
         BNE       X01ERR3       NO, ERROR THEN
* HERE NAME IS OK SO CLEAR IT
         MVC       SYMBOLIC,TKN  SHIFT NAME TO SYMBOLIC
         MVC       #WORK,ZERO    SET LENGTH TO ZERO (NULL)
         $CALL     UJP85ASN      CLEAR NAME AND DEFINE IT IF
*                                IT WASN'T BEFORE
X01GETNX EQU        *
* GET NEXT TOKEN
         GETTKN    TKNCURR       GET NEXT TOKEN
         B         X01LOOP       GO AND STORE R1, CHECK IF A ','
X01ERR2  JOLERR    202,'NAME ''', TKN,''' IS INVALID'
         B         X01GETNX      GET NEXT TOKEN
X01ERR3  MVI       #TKN+1,8      SET LENGTH TO 8 FOR ERROR MESSAGE
         JOLERR    203,'NAME COMMENCING '''# TKN,''' TOO LONG'
         B         X01GETNX      GET NEXT TOKEN
X01END   JOLRETN
         LTORG
         END
```

Figure 6-5. Clearing a List of Symbolic Variables.

Note that this must be LINKED with a name card of **CLEAR** in to the 'JOL.LOAD' Data Set (or any data set that is referenced by the **$$JLOAD** DDCARD).

To invoke the routine, simply code:- **CLEAR** *name-list*;. **CLEAR A,B,C;** will clear symbolic parameters **A**, **B** and **C**.

<u>Example 2</u>: To write an instruction **PROGRAM** or **PROG** that will invoke a user program and set **%LASTCC** to the return code from the program.

NOTE: The **CALL** instruction provided with Jol is functionally similar to this example.

Syntax:-

INVOKE PROGRAM LOAD-MODULE-NAME 'PARAMETERS' ;

```
             COPY      JOLCOM
             USING     JOLCOM,R2
             USING     TKNX,R3
             JOLSAVE   CSECT=UJX02PGM DO SAVE ETC, GET SAVE-AREA
```

```
         GETTKN    3                GET PROGRAM NAME.
         CLI       TKN,C' '         IS THERE ONE?
         BE        X02ERR1          NO, ERROR
         MVC       DBL,TKN          SAVE MODULE OR PROGRAM NAME
         GETTKN    4                GET OPTIONAL PARAMETER
         ATTACH    EPLOC=DBL,DCB=JOLLOAD,ECB=TASKCB,        *
                                PARAM=#TKN,MF=(E,CALLAREA),
         *
                                SF=(E,CALLAREA+4)
         LR        R5,R1            SAVE TCB ADDRESS
         WAIT      ECB=TASKECB
         MVC       TASKRETN(1),X'1D'(R5) SAVE ABEND CODE
         MVC       TASKRETN+1(3),TASKCB+1  AND RETURN CODE
         ST        R5,CALLAREA
         * NOW CHECK FOR ABENDS ETC.
         TM        TASKRETN,128         ABEND?
         BO        X02ABEND
         * HERE IS NORMAL END OF TASK
         MVI       TASKRTN,X'00' CLEAR TOP BYTE OF RETURN
         L         R1,TASKRETN
         CVD       R1,DBL
         UNPK      WORK(4),DBL
         OI        WORK+3,C'0'
X02STRTN  MVC       SYMBOLIC,=CL8'LASTCC'
         $CALL     UJP85ASN         STORE RETURN CODE
         JOLRETN
X02ABEND  JOLERR    402,'TASK ABENDED'
         MVC       WORK(4),=CL5'ABEND'
         MVC       #WORK,=H'5'
         B         X02STRN          RETURN
TASKECB   DC        F'0'
TASKRTN   DC        F'0'
         LTORG
         END
```

Figure 6-6. Example of ATTACHing a Task

Note, the load module must be linked with name **PROGRAM** or **PROG**.

6.5 Adding Instructions Permanently to Jol

Having written and tested your **INVOKE** routines, you can add them permanently to the Jol Compiler load modules. This will result in faster programs as the routines will not have to be loaded from disk each time they are used.

Macro **DEFJOLIN** in the Jol MACLIB contains a list of instructions that Jol considers to be its own - Jol searches for these instructions before loading a macro or an invoke routine.

To add your own instructions, you must do the following:

a) Edit the **DEFJOLIN** instruction, and add your routine name. Note that you add the *common* or user defined name, for example **CLEAR**, and follow it with a number.

For example, if your name is **CLEAR**, and the the module name is **UJP99CLR**, you would code:

(CLEAR,99CLR),

This tells Jol that when the instruction **CLEAR** is found that it must call module **UJP99CLR.**

Note: The module name must start with **UJP.**

b) The instruction you add must be inserted in *alphabetical order* because Jol uses a binary search algorithm to search the table.

c) You must now assemble and link the new table. For example:

```
// EXEC ASMFCL
//SYSIN DD DSN=SYS2.JOL.SOURCE(PJOLINST),DISP=SHR
//LKED.SYSLIN DD *
                   INCLUDE SYSLMOD(UJP00MN)
                   NAME UJP00MN,R
//    DD DSN=&&OBJ,DISP=SHR
//SYSLMOD  DD DSN=SYS2.JOL60.LOAD,DISP=SHR
```

After the process above, your module will be accessible when a User codes the **CLEAR** instruction.

However, the address contained in Register 1 *additionally* points to the following parameter list. The next figure shows the contents of Register 1.

1. X'80',AL3("parm). See Note 1 below.
2. Address of TKNX (same as R3).
3. Address of JOLCOM (same as R2).
4. Address of subroutine UJP9OOP. This routine can be used to output information to the Compiler Phase.
5. Address of UJP85ASN. This routine can be used to assign values to Symbolic Variables.
6. Address of UJP02DCL which may be used to Declare Symbolics.
7. Address of AMACBUF, a special area into which returned instructions are moved (see 14 below).
8. Address of #WORK, a 2000 byte work area for any purpose (but required for Assigns and Declares).
9. Address of DBL, a double-word work area.
10. Address of SYMBOLIC to which names of variables to be DECLARED or have values ASSIGNED to must be moved.

11. Address of JOBDETS (for future use).
12. Address of GENDETS, which contains all the information from the JOLGEN.
13. Address of TOKEN-SPLITTER Routine, which will split any string in #TKNSTRNG into Tokens.
14. Address of UJP980P, the routine which transfers any Jol instruction to the WORKFILE for later interpretation/execution.
15. Address of UJS20REP, the Replace Symbolics Subroutine.
16. Address of UJS21FNC, the Function Subroutine (SUBSTR, TYPE etc).
17. Address of the PRINT Buffer to which information to be printed may be moved (see 18 below).
18. Address of UJSPRINT, the general Print Subroutine (called by JOLPRINT Macro).
19. Five Spare Address Constants.

Figure 6-5. Description of Register 1 Contents.

7 Installing the RACF Exit.

7.1 RACF Passwords and Security Violations

When jobs are submitted to an Operating System with RACF, a **PASSWORD** may have to be coded on the Job Card.

While Jol enables a **PASSWORD** to be placed on the Job Card, this facilities requires the PASSWORD be coded somewhere in the Jol language statements. This may enable others with access to those data sets to examine the PASSWORD, resulting in a security violation.

To overcome the possibility of such a security violation, Jol provides an easily incorporated exit that:

1. Examines the user RACF tables and extracts the PASSWORD.
2. Copies the PASSWORD to the generated JCL, while hiding the PASSWORD in any generated JCL.
3. Enables the Jol SUBMIT Command to be used without the requirement of coding the PASSWORD Parameter, thus passing the **PASSWORD** from job to job.

7.1.1 The Exit

The exit is contained in the Jol Source Assembly data set in members UJG04JOB and UJG05PWD.

UJG05PWD examines the RACF tables, and UJG04JOB outputs the PASSWORD to the generated JCL. The JCL produced is **not** copied the generated JCL compiler listing, even if PJCL is specified.

The JCL interpreter will also supress the PASSWORD, thus hiding it from users.

UJG05PWD may be altered to decrpit the password, if the installation has used an encription technique.

7.1.2 Installing the EXIT

To install the exit, Assemble and Link UJG04JOB with the JCL shown on the following page.

```
// JOB etc
//JOLASM PROC MEM=
// EXEC PGM=ASMBLR
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLIB   DD DSN=SYS1.AMODGEN,DISP=SHR,
//            DCB=BLKSIZE=19040
//         DD DSN=SYS1.MACLIB,DISP=SHR
//         DD DSN=SYS2.JOL60.MACLIB,DISP=SHR
//SYSGO    DD DSN=SYS2.JOL60.OBJ(&MEM),
//            DISP=(MOD,CATLG),UNIT=SYSDA,
//            SPACE=(TRK,(5,5,3)),DCB=BLKSIZE=800
//SYSIN    DD DSN=JOL60.SOURCE(&MEM),DISP=SHR
// PEND
// EXEC JOLASM,MEM=UJG04JOB
// EXEC JOLASM,MEM=UJG05PWD

//LKED EXEC PGM=IEWL,PARM='RENT,REUS,LIST,MAP'
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLIB   DD DSN=SYS2.JOL60.OBJ,DISP=SHR
//SYSLMOD  DD DSN=SYS2.JOL60.LOAD,DISP=SHR
//JOLOLD   DD DSN=SYS2.JOL60.LOAD,DISP=SHR
//SYSLIN   DD DSN=&&OBJ,DISP=SHR
//         DD *
   INCLUDE SYSLIB(UJG04JOB)
   INCLUDE SYSLIB(UJG05PWD)
   INCLUDE JOLOLD(UJG01JOB)
   ENTRY   UJG04JOB
   NAME    UJG01JOB(R)
```

8 Assembler Macro Descriptions

Described in the following pages are Jol Assembler macros that may prove useful when writing Exit or Invoke routines for Jol. These are included in the 'SYS2.JOL60.MACLIB' data set provided with Jol.

8.1 The CLEAR Macro

The **CLEAR** macro is used to clear locations in storage.

A list of locations may be coded, in which case they are cleared according to their type attributes.

If the location is character, it is set to blanks, otherwise it is set to zero.

Examples of the CLEAR macro

```
        CLEAR A

        CLEAR A,B,C,
```

8.2 The IFNULL and IFVALUE Macros

The IFNULL and IFVALUE macros are used to test if a location or a series of locations contain information or if they are blank or zero (depending on the data type).

The IFNULL and IFVALUE macros are coded:

IFNULL *identifier list, label*
IFVALUE

The identifier list may consist of character, binary, hexadecimal and bit locations. The *last name in the list is a label* to which control is passed if the condition is true.

The conditions are tested one after another so that if any are not blank the branch will take place.

Examples

```
    IFNULL LABEL,FIXLABEL

    IFVALUE DDDSNAME,GENDSN
```

8.3 The GETTKN Macro

The GETTKN macro is used to retrieve a token from a statement.

The Jol READ statement routines read an entire statement into a field '**#TKNSTRG**'. Subroutine **UJCTKN** then splits the statement into TOKENS. By use of the GETTKN macro, it is possible to retrieve any token from the #TKNSTRG.

The value returned is a character string of maximum length 253 characters in **#TKN**, a Jol varying string. If the number of the token is out of range (e.g. requesting token 2000), then the length of the string held in #TKN is set to 0, and TKN (the value part) is set to blank.

Register 15 is set to 0 or 4 (if an error occurred).

Register 1 is always set 1 higher than the value requested. For example, if GETTKN 1 was coded, on return register 1 would contain 2, even if an error occurred.

The **GETTKN** macro:

```
                        NO = number
GETTKN        {         REG = register      }
                        LOCN = location
```

or

```
                        location
GETTKN        {         number       }
                        (register)
```

Examples of the GETTKN macro

GETTKN 1

GETTKN (R15)

GETTKN TKNCURR

GETTKN REG=1

8.4 The JOLERR Macro

The **JOLERR** macro generates instructions to print a message or an error.

In general, a parameter list of the type set up for the **#PRINT** macro is created, with the exception that a BAL 14, error routine is generated instead of a LOAD and BALR.

For example:

```
JOLERR 403,'NO MESSAGE TEXT'
```

will generate

```
BAL   14,AUJS134
MVC   0(4,1),=C'CSECT prefix, severity, message number'
MVC   0(4,1),STMT
MVC   0(15,1),= C'NO MESSAGE TEXT'
DC    H'O'
```

Jol Module Naming Conventions

All Jol modules, by convention, begin with UJ.

The next character indicates the purpose of the module, for example **P** for preprocessor modules, **G** for generate and so on.

The next two characters relate to the purpose of the module, e.g. 01 indicates a module concerned with the JOB card, 03 with a data set declaration, or a DD card.

The remaining three characters are used for any purpose, e.g. JOB for UJCO1JOB.

The JOLERR macro uses the *3rd* to the *5th* characters to create a message number - this will identify the CSECT issuing the error message.

The format of the JOLERR macro is:

JOLERR *severity and error number*
, list of identifiers or message texts
[**STMT =** *location*]

Notes

1. **SEVERITY and ERROR numbers**

The severity and error number is made up of a 3 digit number.

The first digit is the severity code and may range from 1 to 5. The severity code is used to distinguish between defaults messages and error messages according to the following table.

Severity	Type of Error
0	Defaults
1	Warnings
2	Errors
3	Severe Errors
4	Terminal Errors
5	Internal Compiler Errors.

All message issued in the Compiler are printed on the list file.

The second two digits of the severity and error number are unique numbers in the CSECT and identify the message. The cross-reference list shows the location of the macro as ER&csect number, and thus by referring to the cross-reference listing the error location may be easily determined.

2. **MESSAGE TEXTS and IDENTIFIERS**

Any identifiers that are to be printed, or message texts are coded separately by commas.

The following are supported :

a) Character strings
b) Varying character strings (determined by the character#)
c) Character string constants
d) Half Word binary numbers.
e) Full Word binary numbers.

The maximum combined lengths of these lists may be 256 characters.

For example the macro

JOLERR 105,'DSNAME''', DSNAME,'''IS INVALID'

will produce the message **DSNAME 'data set name' IS INVALID**

3. **The STMT Keyword**

The STMT defaults to attaching the current contents of the location STMT to the message to identify the statement in error.

It is sometimes desirable to use a different location for the statement number. This may be done by coding STMT = location-containing-the-statement-number.

Examples of the JOLERR macro

JOLERR 501

JOLERR 301,'DSNAME''', DDDSNAME,'''NOT CATLGD: - CATLG FULL’'

8.5 The JOLRETN Macro

The JOLRETN macro is used when it is desired to return to the calling CSECT.

If resets the savearea chain and frees the savearea for other uses.

The default return code is 0.

When a return code of 0 (the default), 4, 8, 12 or 16 is requested, the macro merely generates a BRANCH to a routine which performs the housekeeping required.

When a return code other than the above is requested, a Load Address and Branch is generated.

When a specific registers are requested to be loaded, in line code is generated to return control.

When a register is specified which contains the return code, in line code is generated.

The format of the JOLRETN macro is :

JOLRETN [*registers to be loaded*]**,**
[,**RC=**(*register*)]
[,**RC=***return code*]

Examples of the JOLRETN macro:

JOLRETN

JOLRETN RC = 4

JOLRETN (14,12),RC=8

JOLRETN (14,12)

8.6 The JOLSAVE Macro

The **JOLSAVE** is used to generate a new CSECT and allocate a savearea from a *pool*. The macro generates re-entrant code.

The first time the JOLSAVE macro is issued in an assemble, it sets up equates for RO to R15 for register usage.

Register 15 is destroyed

Register 13 points to a new savearea

The default Base register is R11

The format of the JOLSAVE macro is:
!b

JOLSAVE CSECT=*csect-name*
[**,BASE=***register*]
[**,BASE=(***register1*,*register2***)**]
[,**SIZE=***size-required*]

Examples of the JOLSAVE macro:

JOLSAVE CSECT=UJC01JOB

JOLSAVE CSECT=MAIN,BASE=10

JOLSAVE CSECT=UJGO3DS, BASE=(11,12)

8.7 The #PRINT Macro

The **#PRINT MACRO** is used to edit information to the **SYSPRINT** file.

It automatically converts numeric to character (always dropping the leading zeros). It also starts a new line when the current line is full.

Facilities are provided to:

1. Enter a routine on Head of Form condition

2. Test the current print position, line count and page numbers which are automatically maintained and updated by US37AA.

The format of the #PRINT macro is:

```
#PRINT list of identifiers and/or literal character strings.
                     [,TYPE = DATA (PUT DATA in PL1)]
                     [,SKIP = (0,1,2,3) or PAGE]
```

Any identifiers and literal character strings to be printed are coded seperated by commas.

#PRINT supports following types of data:

a) Character strings
b) Character string constants
c) Varying character strings
d) Half and full word binary numbers

Notes

1. In Jol, a Varying Character is defined as any identifier commencing with the letter '#'. The first two bytes (which must be aligned on a halfword) are assumed to contain the current length of the string. The next bytes contain the information in character format.

2. If you wish to print on the current line, code SKIP = 0 to commence editing into the next available print position.

3. If a line to be printed is greater than the LRECL, wraparound will automatically occur.

4. Numbers are converted to character and leading zeros removed.

Examples of the #PRINT Macro.

```
#PRINT 'HEADING',SKIP=PAGE

#PRINT 'THIS LINE CONTAINS DOUBLE QUOTES'''

#PRINT I,J,DATASET
```

Appendix A : Description of the Token System

8.8 Tokens or Symbols

In the Jol compiler and execute monitor, every statement is read into an area #TKNSTRG which is in TKNX, a DSECT, which is itself based on the area described as TOKENWK1 in the JOLCOM CSECT.

After the statement has been moved to #TKNSTRG, subroutine UJCTKN is called to "split" it into TOKENS or SYMBOLS.

When a statement is "split" into TOKENS, the 512 full words described as TKNDESC are set up in such a manner that when the GETTKN macro is issued - calling on subroutine UJCGETKN - the word or token required can be quickly obtained and transferred to #TKN, and the type (numeric, character, etc.) moved to TKNTYPE.

The statement is 'split' by analysing it from left to right and describing each token in successive full words in the word list. The 'description' contains the length of the token, its type and its offset within the statement.

For example, if a statement was

DCL X DS JOL. MACROS;

then the table would resemble:

Length	**Type**	**Offset**	**Token**	**Number**
3	2	0	DCL	1
1	2	5	X	2
2	2	7	DS	3
3	2	10	JOL	4
1	1	13	.	5
6	2	14	MACROS	6
1	1	20	;	7

Blanks are ignored by UJCTKN unless they are embedded in quotes, and serve as delimiters only.

By coding GETTKN 1, TKN would be set up as DCL with #TKN equal to length 3 and TKNTYPE set to 2 (meaning identifiers). The remainder of TKN is cleared to blanks. On return from the GETTKN macro register 1 contains the token number of the token following the one requested.

Note : These routines are free standing and have no dependencies on any part of Jol.

They are available on a CBT tape.

Readers Comment Form

This manual is part of the Jol library that serves as a reference source for Managers, Systems Analysts, Programmers and Operators. This form may be used to communicate your views about this publication.

Clarke Computer Software may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use any information you supply.

Possible topics for comments are:

Clarity Accuracy Completeness Organization Legibility

If comments apply to a Selectable Unit, please supply the name of the Selectable Unit _______________.

If you wish a reply, give your name and address

____

____

____

____

____

Number of latest Newsletter associated with this publication: _____________

Please send your comments to:

Clarke Computer Software,
50/45 Riversdale Road,
Hawthorn,
Victoria,
AUSTRALIA, 3122.

Email: clementclarke@ozemail.com.au