Scheduling and Networking Guide

Jol

Universal Command Language

Version 5.1

SCHEDULING AND NETWORKING GUIDE

Open System Command and Retrieval (OSCAR).

Jol Command Language " **Scheduling and Networking Guide**".

Sixth Edition (November, 1999)

This is a major revision of, and obseletes, the
Fifth Edition of the Jol Command Language
" **Scheduling and Networking Guide** " September 1987

Clarke Computer Software,
Ground Floor,
45 Riversdale Road
Hawthorn,
Victoria
AUSTRALIA, 3122

Telephone + 61 (0)401 054 155
Email clementclarke@ozemail.com.au

Preface

This publication provides a general description of the Jol Universal Command Language Scheduling and Networking System. It is intended for those making use of the Jol Scheduling System, using either Jol or JCL. You should also be familiar with the *Jol Concepts and Facilities Manual* and the *Jol General Information Manual*.

Although the Jol Scheduling and Job Networking facility is equally at home with Jol or JCL, conversion to Jol should be seriously considered whenever an application is being modified because of maintenance or enhancement. In addition, because of Jol's program structure future modifications to the system will be made much easier.

> ***Note:*** It is not necessary to convert completely to Jol. Jol will work in a mixed environment. Furthermore, Jol's powerful logic facilities and symbolic variables can be used with your own system's JCL to provide, for example, automatic submission of JCL based on the date, time and so on.

The book is divided into the following chapters:

- Chapter 1, *Introduction*, introduces the Jol Universal Command Language, and briefly describes how to submit existing JCL with Jol, and extra facilities available when using Jol with JCL.

- Chapter 2, *Job Networks,* and chapter 3, *Dependent Jobs*, describe how to submit other jobs from currently executing jobs, and how to pass variable symbolic variable data from one program to another.

- Chapter 4, *Scheduling*, describes the Jol automatic scheduling facility, and how to set up **HOLIDAY** details.

- Chapter 5, *Implementation*, describes the data sets and members of those data sets necessary to implement the *Jol Scheduling and Networking Package*.

Recent Changes to the Jol Universal Command Language

Scheduling and Networking Facilities

- *Jobs may be defined as part of a network, and automatically submitted on certain days.*

TSO Support

- *Options to allow the job to execute under TSO or Background, using the same Command Language.*

ALLOCATE, OPEN, READ and WRITE Instructions

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

Associated Documentation

- ***Jol Reference Manual and Jol Reference Guide***
- ***Jol Concepts and Facilities Manual***
- ***Jol General Information Manual***
- ***Jol Answers to Questions***
- ***Converting to Jol***
- ***Jol Evaluation Plan and Financial Work Sheets***
- ***Jol System Programmer Guide***
- ***Jol Program Logic Manual***

.

Contents

Preface **3**
Recent Changes to the Jol Universal Command Language 4
Associated Documentation 4

Introduction **7**
Dependent Job Networks 7
Jol can Submit Jobs and Control Concurrent Execution of Jobs 7
Passing Symbolic Variables from Job to Job 7
Jol Panel Facility Eases Writing Panels 7
Automatic Submission of Jobs on Predefined Days 7
Using Jol to Submit Existing JCL 8
An Overview of the Jol Universal Command Language 11
Main Features of Jol 11
Areas of Use 12
Jol is a general purpose language 13
Jol Instructions 13
Creating New Commands for Your Installation 15
FACILITIES 16
Calendar 16
Testing Symbolic Variables and Return Codes 16
New Data Sets 17
Operator Communication 17
Preprocessing 17
Text 17
The INCLUDE Library 19
Dependent 19
Jobs 19
Other Facilities 19
Procedural 19
Programming 20
SUMMARY 20

Job Networks **22**
Job Networks 23
Submitting Operating System JCL Jobs 26
Submitting Dependent Jobs 27
Examining the Status of a Network 27
Restarting a Network 28
JOL * 28
Building in a Restart Job 28
Forcing Jol to Examine the Network Status Early 28
Method of Operation 29

Dependent Jobs **30**
Dependent Job Networks 30
Dependent Job Processing 30
Passing Symbolic Variables and Submitting Other Jobs 31
Steps Required to Create the Jobstream 33

Scheduling Jobs by Date **35**
Scheduling 35
Rules 36
Planning Ahead 36

How the Schedule Facility Starts the Required Jobs 38
Catering for Special Requirements not Handled by Schedule 38
Telling Jol Which Days are HOLIDAYS 40
Other Useful Facilities for Use in Scheduling 41

Implementation 42
Language 42
Data Sets Required *for NETWORKING* 42
Data Sets Required for SCHEDULING 42
Method of Operation 42
The NETWORK Command 44
The SUBMIT Command 44
The ENDNET Command 44
The CHECKNET Command 44
The NETSUB Command 44
Documented Example 44

Dependent Job Networks

All large computer systems provide a method of running several programs in a predefined sequence. You can also test if a program executed successfully before allowing other programs to run. Such a sequence of programs is usually referred to as a *job*.

However, only a few Operating Systems allow you to run *entire jobs* in a predefined sequence, or in *parallel with other work in a controlled environment.* IBM's JES3 is one such system, and Fujitsu's FSP/X8 is another. Without a controlled environment, Operations Personel are required to submit jobs when one ends, and to control the execution of jobs concurrently. These days with extremely large computer systems running hundreds, if not thousands, of jobs per day, this places unrealistic demands on the Operations Personel, and mistakes are often made.

Jobs are usually broken up into units of work that run for half an hour, an hour, etc. The reasons are many - for example it is easier to restart a smaller job than a large one if the job fails, maintenance is easier with smaller jobs, and so on.

Jol can Submit Jobs and Control Concurrent Execution of Jobs

The Jol Dependent Job Networking facility operates on all systems on which Jol itself operates. It can automatically submit your own Operating Systems Job Control Language statements (the JCL), or you can use Jol Universal Command Language jobs.

Passing Symbolic Variables from Job to Job

An added feature is that Symbolic Variables can be passed from job to job. Additionally, Symbolic Variables can be saved in a Partitioned Data Set, and retrieved by any Jol or JCL job running under Jol.

Jol Panel Facility Eases Writing Panels

The Jol **PANEL** instruction allows you to enter Symbolic Variables that can be validated with the Jol **IF** instruction. These may be saved in a data set, or used directly to either generate Jol or JCL Job Streams.

Automatic Submission of Jobs on Predefined Days

In addition to automatically submitting jobs in a predefined sequence for either *sequential* or *concurrent* execution, Jol will also submit jobs on particular days

or dates. Again, these jobs can be written in Jol Universal Command Language Statements, or your native Job Control Language.

Facilities are provided to start work (or jobs) on particular days (such as June 30th, 1988), on certain days (such as every Monday, or on Holidays), or the day(s) before or after days you specify. An easily modifiable table of Holidays is provided with the system. How to use these facilities is described in later chapters of this book.

Using Jol to Submit Existing JCL

Notice that you do not have to convert your existing JCL to run under the control of Jol. Jol provides the facility to use your existing JCL while taking advantage of Jol's powerful logic. The code below shows the use of Jol logic to alter variables dependent on certain conditions and the submission of the job. See figure **1-1** below for an example.

```
Statement                Jol Code
Number

      1     IF %DAY='FRIDAY'
                  THEN DO ;                  /* Weekly Processing */
      2           %PROC='WKPROC' ;
      3           %SPACE='(CYL,10) ;
      4     END ;
      5     ELSE DO ;
      6           %PROC='DAYPROC' ;          /* Daily procedure */
      7           %SPACE='(CYL,5)' ;
      8     END ;
      9     SUBMIT '//JOB1 JOB etc.....'
     10           '//STEP1 EXEC %PROC, '
     11           '// SPACE="%SPACE"' ;
```

Explanation:

Statement 1 — Uses Jol's built in calendar to determine the day of the week. If the day is Friday it initiates the **DO** group (statement 2).

Statement 2 — Sets the value of the variable **'%PROC'** if today is Friday.

Statement 3 — Sets the value of the variable **'%SPACE'** if today is Friday.

Statement 4 — **END**s the **DO** group that was started by statement 1.

Statement 5 — Initiates another **DO** group if today is not Friday.

Statements 6 and 7 — Set the values for the variables **%PROC** and **%SPACE**.

Statement 8 — **END**s the **DO** group started by statement 5.

Statements 9, 10 and 11 — Instructs Jol to **SUBMIT** the job to the JCL queue after substituting the symbolic values in the JCL.

Therefore, if today is Friday, then the JCL to be submitted would be:

```
//JOB1 JOB....etc
//STEP1 EXEC WKPROC,
```

```
//    SPACE='(CYL,10)'
```

otherwise, the JCL submitted would be:

```
//JOB1 JOB....etc
//STEP1 EXEC DAYPROC,
//    SPACE='(CYL,5)'
```

Figure 1: Using Jol to Submit JCL Example (1).

Another method of submitting the job above is shown below.

```
Statement
 Number          Jol Code

     1    IF %DAY='FRIDAY'
          THEN DO ;      /* Weekly Processing */
     2         SUBMIT         '//JOB1 JOB etc.....'
               '//STEP1 EXEC WKPROC,'
               '// SPACE="(CYL,10)" ;
     3    END ;
     4    ELSE DO ;
     5         SUBMIT         '//JOB1 JOB etc.....'
               '//STEP1 EXEC DAYPROC,'
               '// SPACE="(CYL,5)" ;
     6    END ;
```

Explanation:

Statement 1 — Checks Jol's calendar to determine if the day is Friday. If it is **FRIDAY,** it initiates the **DO** group (statement 2).

Statement 2 — **SUBMIT**s the following JCL on Fridays:

```
//JOB1 JOB....etc
//STEP1 EXEC WKPROC,
//    SPACE='(CYL,10)'
```

Statement 3 — **END**s the **DO** group that was started by statement 1.

Statement 4 — Initiates another **DO** group if today is not Friday.

Statement 5 — **SUBMIT**s the following JCL on days *other than* Fridays:

```
//JOB1 JOB....etc
//STEP1 EXEC DAYPROC,
//    SPACE='(CYL,5)'
```

Statement 6 — **END**s the **DO** group started by statement 5.

Figure 2: Using Jol to Submit JCL Example (2).

An Overview of the Jol Universal Command Language

A Command Language is the highest level of communication between you and the computer. Command languages tell the computer what to do, when to do it, and what to do with the result. Programming Languages, such as PL/I and COBOL, give the computer detailed instructions on how to do it.

Main Features of Jol

Some of Jol's *main* features include:

- Commands are coded in Jol, which is a free format English-like language similiar to TSO CLISTs, PL/I, Pascal and "C".

- You can specify whether the job is to run *immediately* (under TSO), or in a *background* region, either with or without JCL, or under full control of a monitor which uses Dynamic Allocation instead of JCL.

  The same command language can operate both in *foreground* and *background*.

Other features of Jol include:

- Jol is easy to use.

- Jol has facilities for scheduling jobs automatically, depending on the day of the week, particular dates, or only on working days.

  Both Jol and JCL jobs may be used.

- Jol has an extremely comprehensive Macro language that enables you to write your own Jol commands tailored specifically to your installation.

- Jol contains **IF, THEN, ELSE, AND**/**OR,** and **DO** logic which provides the Programmer with a powerful tool for manipulating programs in jobstreams based on variable conditions, calendar information, or **ABEND** situations.

- Jol allows you to define, initialize, test, branch and perform arithmetic on Symbolic variables. Hence, you can create data set names, programs and instructions depending on symbolic variable or parameter information.

- Jol allows you to **ALLOCATE**, **READ** and **WRITE** to data sets during the compilation. Therefore, you can access data, and dynamically alter your job depending on other data on your computer.

- Jol allows you to simply write full screen data entry panels.

- Jol performs thorough error detection, that includes a semi-simulation

phase and catalog searching to ensure that the job will execute correctly.

- Jol has many User Exits available that allow installation standards to be enforced. At the same time, changes can be made to a User's job before it is executed.

In other words, Jol fully complements IBM's JCL, CLISTs and SPF in one very easy to use and learn Command Language.

Areas of Use

There are **four** fairly broad areas that you, as a computer User, can use Jol in, and these areas often overlap.

1 Scientific

Scientific Users tend to set up procedures that work - and then never change them again. For example, a **Fortran** Program may have been written by yourself or someone else and you want to execute it, with some sort of data file, and printed output.

2 Programming

Programmers generally know a considerable amount about the inner workings of the machine, and want to execute quite sophisticated programs of your own design, together with Copies, Compiles and so on.

Usually your jobs will be short and often will contain many of the functions of 3 below.

Your Commands to the system will often change, and so you need a **Dynamic Control Language.**

3 Production Control

Production Control Experts are in charge of writing all the Production job control procedures from details supplied by Programmers.

Often these procedures can be extremely complex and require really sophisticated use of the machine in order to have the work run through the System in the shortest possible time.

Generally, you will have fairly static procedures, but they will be complex and well thought out, and often contain instructions that should only be executed if some sort of error condition arises. For example, if a data set cannot be read because it has been damaged, you must supply some method for recreating the data set.

Jol also has facilities for scheduling jobs automatically, depending on the day of the week, particular dates, or only on working days.

4 Data Management

Data Managers are in charge of general housekeeping procedures, and therefore be responsible for copying data sets from one volume to another, deleting spurious data sets, and so on. Furthermore, some of the functions may be dependent on the data (or some other variable), and you would like to be able to write a procedure that can be executed on a daily basis, but perform different work depending on the date.

And so your use of the machine will fall into two main areas, a static one where, say every Tuesday you are required to copy data sets for Security Purposes, and a very dynamic area where you are required to delete data sets, or recreate them at a moment's notice.

Jol is a general purpose language

Jol is a Command Language which covers all the above applications in simple **English-like** statements. These instructions are discussed in the *Jol Concepts and Facilities Manual* and the *Jol General Information Manual*. Full details of the language can be found in the *Jol Reference Manuals* and *Guides*. If you are a Programmer most of the instructions provided with Jol will be similar to the ones you are already familiar with in programming languages.

Jol Instructions

The following is a partial list of instructions included in **Jol**. Most of these instructions can be used to enhance your use of JCL procedures too. For example, the Jol PANEL instruction can be used to implement a menu system - data entered by Users can then modify your own JCL procedures before the jobs are submitted. Similarly, **ALLOCATE, READ** and **WRITE** instructions can be used effectively to tailor your existing JCL for submission. Some of the instructions described below are only available when using Jol in "native mode" - when using Jol to submit and alter your pre-written JCL, macros such as **_SORT**, **COPY** and **RUN** are not applicable.

Most of the Jol instructions listed below manipulate **DECLARED** variables, data set identifiers and load modules while others provide information to the terminal User or the Operator. These instructions, and others, are covered in full in the *Jol Reference Manual* and *Jol Reference Guide*.

Instructions	Use
ALLOCATE	A Data Set
ASSIGN	A Value to a Symbolic Location
BUILDGDG	Build Catalog Entries for GDGs
CATALOG	A Data Set Identifier or Data Set
COPY	A Data Set or DSID to another
DELETE	A Data Set Identifier or Data Set
DISPLAY	A Message on the System Log
DO	A Group of Instructions
EDIT	Symbolic Variables
END	A Group of Instructions
IF	Test the Value of a Return Code or Symbolic

	Parameter
INCLUDE	Source Text from a Library
INVOKE	A Load Module into the Jol Compiler
KEEP	A Data Set Identifier or Data Set
LIST	Data Sets
LISTCAT	List the Catalog
MERGE	Data Sets
OPENFILE	Open a File for Input or Output
PANEL	Create a Formatted screen for data entry
PRINT	Print Data Sets
READ	A Record from either~ a File or the Terminal
RUN	A load Module or Program
SCRATCH	A Data Set Identifier or Data Set
SIGNAL	An Error or Warning
SORT	A Data Set or DSID to another
START AT	Start At a Specific Part of the Program
STOP	The Execution of the Job
STOP AT	Stop At a Specific Part of the Program
STOP WHEN	Stop When Specific Return Codes are Found
SUBMIT	Other Jobs
TEXIST	Test if a Data Set Exists for Scheduling or Restarts
TYPE	A Message on the Operator's Console
UNCATALOG	A Data Set Identifier or Data Set
WRITE	A Record to a File or the Terminal

Figure 3. Partial List of Jol Instructions.

DECLARE or **DEFINE** statements are used to describe **Data Sets** and **Programs**, and to set up **Symbolic Variables.**

A **JOB statement** gives details such as the name of your job and its expected elapsed and **CPU** times.

A **MACRO statement** defines macro commands and sets up defaults for keywords used in the macro commands.

As shown in the table above, some of the functions Jol can perform are:

- **RUN** programs.
- **PRINT** data sets.
- **COPY** data sets or volumes containing data.
- **CATALOG** or **DELETE** data sets.
- **Test** Return Codes, Error conditions and Symbolic Variables.
- **SUBMIT** other jobs to the system.
- **ALLOCATE, READ** and **WRITE** data sets.
- **Examine** the System year, month, day and date.
- **List** Catalogs.
- **Write** your own new Jol Instructions.
- and more.

For example, to **PRINT** a data set you could specify:

```
PRINT JOL.INCLUDE(PAYROLL);
```

This will print on the printer member **PAYROLL** of the **JOL.INCLUDE** data set.

To **SORT** a card image data set and **CATALOG** and **PRINT** the sorted output file may be coded as:

```
DCL CARDS *;
cards
EOF;
DCL OUTPUT DATA SET          /* Define output data set */
    SORTED.CARDS             /* Define name */
    FB 80,800                /* Define record format */
    100 RECORDS              /* Define space */
    SYSDA;                   /* Define unit */

SORTSTEP:
    SORT CARDS TO OUTPUT     /* Sort cards as specified*/
        FIELDS=(10,10,CH,A);         /* over fields */

    IF SORTSTEP=0            /* If sorted successfully */
    THEN DO;
        CATLG OUTPUT;        /* then catalog data set */
        PRINT OUTPUT;        /* and print it. */
    END;
```

Figure 4: Small Sort Example.

Creating New Commands for Your Installation

Jol commands can be combined with themselves and with any other program to form new commands tailored specifically to your installation. With Jol you can also execute commands from within commands, adding greatly to the flexibility and simplicity of procedures. This open-endedness is one of the highlights of Jol.

The Jol Macro Language may be considered as a major extension of the **JCL PROC**edure statement, and provides a means of adding new instructions to the Language quickly and easily, thus allowing the tailoring of Jol to your specific installation.

The Macro language provides you with the ability of using *any* of the instructions *above*, or instructions *you develop*, and combining them into new instructions specifically for your installation, or a specific application.

The **SORT macro,** which is a standard Jol **macro,** is called with parameters and allocates work areas for the sort program and performs the sort function for you.

Macros are recursive and reusable.

For example, to make a command to both Print and Delete the input data set may be coded as:

```
PRDEL : MACRO;
     PRINT %LIST(1);
     DELETE %LIST (1);
END;
```

See writing **Macro Commands** in the Jol Reference Manuals for more details.

FACILITIES

Calendar

Before any statements are read, Jol initializes the following **Symbolic Variables** that may be used to start jobs or parts of jobs on particular days or dates.

%SYSUID	**:**	**the System User Identification**
%SYSDATE	**:**	**the current date, for example 85200**
%DAY	**:**	**MONDAY, TUESDAY etc**
%MONTH	**:**	**JANUARY, FEBRUARY etc**
%MONTHNO	**:**	**01,02 through to 12**
%DAYNO	**:**	**01 - 31**
%YEAR	**:**	**1988. 1989, ...**
%HOURS	**:**	**0 - 23**
%MINS	**:**	**0 - 59**
%SECS	**:**	**0 - 59**
%SYSTEM	**:**	**MVS, F4, DOS, X8, VS1 or VS2**
%SPOOL	**:**	**HASP, ASP, JES1, JES2, JES3 or blank**

Figure 5: Pre-initialized Symbolic Variables.

Use of these variables, in combination with a list of jobs you provide to run on certain days, provides an automatic scheduling facility. This is described in detail later.

Testing Symbolic Variables and Return Codes

A simple **IF** statement allows you to test the values that your programs returned and the values of any symbolic variables. For example:

```
IF MYPROG = 16
THEN STOP 'ERROR OCCURRED';
```

New Data Sets

Operator Communication

Other statements allow you to **TYPE** a message on the operator's console, or merely **DISPLAY** a message on the system log for your purposes. For example:

```
TYPE 'THIS MESSAGE APPEARS ON THE OPERATOR'S CONSOLE';
```

Preprocessing Text

You may alter your source program as Jol is compiling your program into executable instructions for the Operating System. This is known as preprocessing the source text. You can:

- Define Symbolic Variables and initialize them.
- Test and change the current contents of Symbolic Variables.
- Perform Arithmetic and String operations on Symbolic Variables.
- Replace text (even in card image files) with Symbolic Variables.
- Save and Restore Symbolic Variables from disk data sets.
- Modify, alter and/or insert program statements. You can indicate which sections of the Source Program are to be compiled.
- Allocate, read and write data sets. The data read can then modify the program as above.
- Unconditionally or conditionally **INCLUDE** Jol Statements or card image files from libraries.
- Conditionally compilate your program based on the values of Symbolic Variables.
- User written Assembler, Cobol or PL/I code can be executed and generate Jol statements.
- Write full restart instructions into your program, these being executed only when you wish to restart your job after a failure.
- Write procedures that contain instructions for, say, daily, weekly, monthly and yearly processing, but only execute those parts of the procedure that are required for a particular run by setting a symbolic variable to a particular value, testing it and generating different instructions according to its value.
- Read data from an external data set with the **GET** and generate different instructions depending on the data that was read.
- **INVOKE** any ordinary problem programs or specially written programs into the **Compiler** itself, and do any normal program function (read data sets and so on), and generate different instructions according to what functions the program performed.

- Write special programs that become extensions to the Jol language with the **INVOKE** facility.

- and more. . .

You can freely intermix preprocessor and macro statements with the rest of your instructions. Preprocessor and macro command statements are executed when they are encountered in the source text. The section *Compile Time Facilities* in the *General Information Manual* discusses this more fully.

The INCLUDE Library

So that procedures may be broken up into small logical units, you may store all or part of your procedures in a library and **INCLUDE** it as part of the input to Jol at compile time. The use of **Symbolic Variables** allows you to make temporary changes to the procedure easily; so does overriding.

There are a number of ways in which this feature may be used. Production jobs or work is usually stored in a member of the **JOL.INCLUDE** library. When it must execute, the member is simply included, and all the instructions in the member are compiled and executed. Another use is in testing a series of related programs. You could place all the data sets and program definitions in a library and input (either on cards or with a main controlling module) only those **RUN**s, **KEEP**s, **SORT**s and other instructions that you wish to execute this particular run. Thus a number of different Programmers can share information about a number of data sets and programs.

Dependent Jobs

There are two main methods provided to simplify the control and submission of Dependent Jobs.

The first method uses the **SUBMIT** Macro Command to execute Jol itself from within your job at execution time.

The second method is to set up a Jol Network using the **NETWORK** instruction.

Both methods permit you to execute dependent jobs one after another without operator intervention, thus ensuring that jobs are run in the correct sequence.

You can:

- Submit more than one Jol program
- Define Jol compiler print options
- Pass symbolic data from **Job** to **Job** automatically.

Other Facilities

Other facilities include:

- Program registration. When programs are **REGISTER**ed, Jol knows the *language* the program is written in, the *filenames* it uses, and other details. When a registered program is compiled, Jol automatically calls the correct language translator. When a registered program is **RUN**, Jol automatically binds *filenames* or *ddnames* with data set names.
- Centralized data set description facilities.
- Access to data sets, such as machine schedules. Data can be read into or written from *Symbolic Variables*, tested, and jobs *(or parts of jobs)* submitted based on that data, the date or both. Symbolic Variables can be passed from job to job.
- Other instructions to assist with day to day operations.

Procedural

Jol programs are written in a procedural format that is already familiar to Programmers using programming languages such as COBOL, PASCAL or

Programming

PL/I. The procedural format provides you the flexibility to solve the most complex type of requirements in a logical straightforward manner.

In **COBOL** or **PL/I**, you **DEFINE** or **DECLARE** variables and then use instructions to alter those variables. In Jol, you also **DEFINE** or **DECLARE** Data Set and Program details which *may* be required for your job and then you use simple instructions such as **RUN** and **SORT** to set the job in motion. In Jol:

- *Data Set* declarations specify *record-format, volume, unit, space* and other information.
- *Program* declarations specify whether the program's internal filenames (**DDNAMES**) *read*, *write*, *update* or *extend* the data set variables previously described.
- *Instructions* such as **RUN, IF ... THEN, COPY, SORT,** and **PRINT** manipulate these variables as you would variables in other high level languages. Other simple instructions are provided to **test** errors or conditions, **submit** jobs and so on.

SUMMARY

Jol statements and declarations are English-like, easy to write and understand. Whereas most computer languages are relatively rigid in their syntax and coding conventions, Jol has a free format and in many cases keywords are entirely optional; as are equal signs and brackets. Where one blank may be coded, any number may be used, and comments may be used freely for annotation.

To summarize, Jol has English-like instructions to:-

- Define your programs and data files.
- Execute your programs.
- Run your job in batch or interactively.
- Test the values the programs returned and the values of symbolic variables.
- Keep, catalog, scratch, delete or uncatalog data sets.
- Copy, merge, sort, print, list (and so on) data sets.
- Submit jobs at any time from an executing job, while passing symbolic parameters from job to job.
- Allocate, read and write data to a file, and optionally submit jobs or parts of jobs based on that data.
- Use the calendar to submit jobs or run parts of any job at any specified time on any specified day.
- Write new commands with the Macro and Invoke facilities.

- Communicate with the Terminal User.
- Communicate with the Operator.
- Alter the program at compile time.
- Stop processing at any time.

Job Networks

This section of the manual discusses one method you can use to chain together many jobs to run in a particular sequence. This chapter shows how you can have Jol manage a network and **SUBMIT** a job or jobs after a particular job or group of parallel running jobs has ended. The next chapter demonstrates how you can **SUBMIT** a job from a currently executing job (**Dependent Job Processing**).

The major differences between the two methods are:

- With *Job Networking*, Jol manages the submission of jobs and controls *parallel* execution of jobs and the waiting for the termination of jobs before submitting other jobs in the network. Basically, you code your Jol programs as though they are independent jobs, and then code a special set of Jol network instructions that then submit the jobs in the desired sequence for you.

  Jobs are usually submitted only when the current job ends, although you can force Jol to examine the status of the network at any time, and depending on the status, submit other jobs.

- *Dependent job processing* allows you to **SUBMIT** a job at any time from the current job. You code the **SUBMIT** instructions at appropriate points in your Jol program. The **SUBMIT** commands can be coded before the job ends, and allows for parallel processing, but you cannot wait for two jobs to terminate before submitting another.

It is recommended that you be familiar with, or review, the **SUBMIT** and **SAVESYMS** commands. These commands, with their protocols, are explained in the *Jol Reference Manual* and *Jol Reference Guide*, and in the Appendices of this book.

Job Networks

Often it is desirable to process two or more jobs in parallel, and after those jobs have reached completion, start one or more jobs.

To prepare a Jol network is a simple task. You can use your previous Jol or JCL jobs without alteration. You prepare your network commands and place them in the **JOL.NETWORK** partitioned data set under the member name of the network.

A typical network job might be that jobs **JOB1** and **JOB2** are to run, and after both jobs have completed **JOB3** is to commence. This can be shown as:

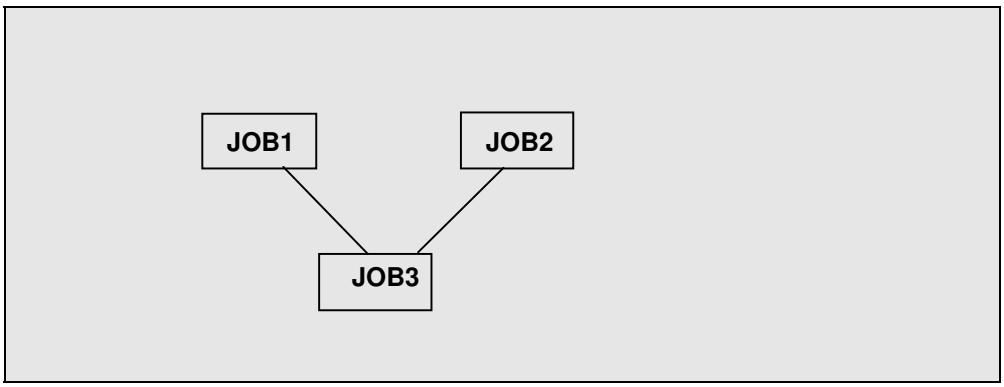

Figure 6: A Small Job Network

This may be coded in Jol as shown on the following page:

```
NETWORK ONE;
   SUBMIT JOB1;
   SUBMIT JOB2;
   SUBMIT JOB3 AFTER JOB1 & JOB2 ENDED;
ENDNET;
```

Notice the Jol instructions **NETWORK** and **ENDNET**. The **NETWORK** instruction is followed by the *name* of a network. This defines a unique network so that you can submit the same jobs in different networks, and still allow Jol to network the correct jobs. The **ENDNET** informs Jol that any instructions following are not part of the network.

You can also use other Jol instructions such as **PANEL** in a network. Any instruction will be executed as normal, except for **SUBMIT**, which is copied to a work file for possible re-execution at the end of each job.

For example, the jobs above may require the Operator to provide Symbolic Variables for the jobs. For example:

```
PANEL ('ENTER TODAY''S DOLLAR RATE',DOLLVAL,7);


NETWORK ONE;
   SUBMIT JOB1 SYMS('DOLLVAL=%DOLLVAL);
   SUBMIT JOB2;

   SUBMIT JOB3 SYMS('DAY=%DAY')
            AFTER JOB1 & JOB2 ENDED ;

ENDNET;
```

Figure 7: A Small Network with PANEL and Symbolic Variables.

A more complex example is represented below:

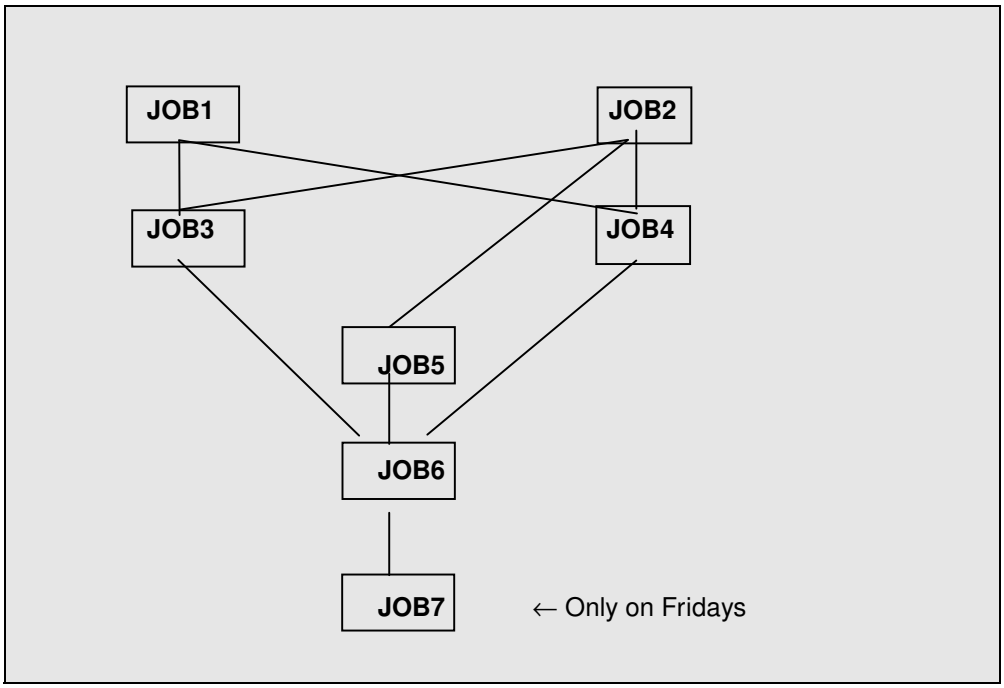

Figure 8: An Advanced Network.

In this example, **JOB1** and **JOB2** may proceed in parallel. The other jobs must execute in the following sequence:

- **JOB3** can commence processing after *either* **JOB1** or **JOB2** have ended.
- **JOB4** cannot commence until *both* **JOB1** and **JOB2** have finished.
- **JOB5** must wait for **JOB2** to end before it can start.
- **JOB6** must wait for **JOB3, JOB4** *and* **JOB6** before it can start.
- **JOB7** must wait for **JOB6** to end, and it is to be executed *only on Fridays*.

One possible method of coding the example above is presented:

```
NETWORK TWO;
     SUBMIT JOB1;
     SUBMIT JOB2;
     SUBMIT JOB3 AFTER JOB1 OR JOB2 ENDED;
     SUBMIT JOB4 AFTER JOB1 AND JOB2 ENDED;
     SUBMIT JOB5 AFTER JOB2 ENDED;
     SUBMIT JOB6
          AFTER JOB3 & JOB4 & JOB5 ENDED;
     IF %DAY='FRIDAY' THEN
          SUBMIT JOB7 AFTER JOB6 ENDED;
ENDNET;
```

Figure 9: Coding the Advanced Network.

Submitting Operating System JCL Jobs

The Jol **SUBMIT** command is extremely flexible: you can pass symbolic variables from job to job, test completion codes before submitting other jobs and other facilities are available.

You can also use **SUBMIT** to submit JCL jobs by enclosing the JCL to be submitted in quotes. When **SUBMIT** detects that parameters have been coded in quotes, it passes them directly through to the Operating System, *after replacing any Jol Symbolic Variables*. For example:

```
NETWORK TWO;

     Job1:
          SUBMIT          '//JOB1  JOB (ACCT),NAME'
                          '// EXEC VALIDATE,'
                          '//      PARM="PROG1=%DAY"';

     Job2:
          SUBMIT          '//JOB2  JOB (ACCT),NAME'
                          '// EXEC UPDATE'
                          AFTER JOB1 ENDED;
     SUBMIT JOB3 AFTER JOB1 AND JOB2 ENDED;
ENDNET;
```

Figure 10: Submitting JCL in a Network.

Notes

Note that when you are submitting JCL or MS/DOS commands, you must code a Jol label (**Job1:** in the example above) so that Jol can identify the job in the Network.

Submitting Dependent Jobs

In addition to the Jol **NETWORK** instructions, the Jol **SUBMIT** Command ensures that the correct order of processing interdependent jobs is observed. Other jobs may be **SUBMIT**ted at any time - even before the end of the current job.

Usually, the source text for **SUBMIT**ted jobs can be found in a library.

Please note that ASP and JES3 have facilities allowing the creation of a "Job Net". Jobnets are not yet supported by Jol.

Notes

- No job is submitted more than once, even if there are two or more **SUBMIT**s for it. This allows you to code multiple conditions for a job to start.
- You must code the jobs that must run *before* the successor jobs. However, jobs are not necessarily submitted in the order that they are coded - rather they are submitted as soon as Jol can determine that the dependencies have been satisfied.
- Jobs in the network may also **SUBMIT** other jobs. However, these jobs, which may run in parallel with any jobs in the network, are not executed under control of the network manager.
- If an abend occurs in a job in the network, the default action is to stop the network. The **IF ERROR** instruction can be used to determine if an abend occured, and any appropriate action can be taken.

Examining the Status of a Network

The Jol Network Facility uses a *single standard* Partitioned Data Set to contain information about *all* the Networks under its control. To review the status of any job in a Network, you can use ISPF or PFD to examine the data set **SYS2.JOLNET.STATUS**.

Each member of the **SYS2.JOLNET.STATUS** data set is the name of a Network, and each member contains the current state of each job in the Network as a series of assignment statements for Jol. For example, in a network called **PAYROLL** there may be jobs **VALIDATE** and **UPDATE**. The member **PAYROLL** will contain statements similar to:

```
%VALIDATE='ENDED';
%UPDATE='SUBMITTED';
```

Restarting a Network

Should a Networked job fail for any reason, correct the problem, and enter:

```
Start Jol with:
        JOL *

Then enter

        $JOB;          /* Create a Job Card */
        CHECKNET network-name;
/*
```

This will restart the job net for you.

If you must restart at an earlier job in the network, simply edit the **SYS2.JOLNET.STATUS**(*network-name*) data set and alter the network to the status you require, and follow the instructions above.

Building in a Restart Job

Often, you can program restarts for various jobs or networks if a failure occurs. To do this, code similar to the following may be used:

```
IF ERROR          /* An ABEND Occurred */
| MAXCC>4         /* A Program Detected an Error */
THEN DO;
     SUBMIT RESTORE;  /* Restore Files */
     STOPNET;         /* Stop Current Network */
END;
```

The job called **RESTORE** can restore the data sets, and restart the Network.

Forcing Jol to Examine the Network Status Early

Usually, Jol will wait until the end of a job before it issues a **CHECKNET** instruction. You may decide that halfway through a job, you would like Jol to examine the Network, and start other jobs if it is able to. To do this, simply code a **CHECKNET** instruction at the appropriate point if the job. For example:

```
        IF MAXCC<4          /* Test for errors first */
        THEN CHECKNET;
```

Method of Operation

When Jol finds a **NETWORK** command, it does the following:

- It opens a special work data set into which the **SUBMIT** command will copy information. This is checked at the end of every job in the network.
- It sets an indicator that is used by the **SUBMIT** command.

When a **SUBMIT** command is executed, it does the following:

- If the indicator mentioned above is not set, the **SUBMIT** instruction operates as usual. That is, the job is submitted to the operating system for execution.

Otherwise, the **SUBMIT** takes special actions for controlling the network.

- If the **SUBMIT** is an unconditional **SUBMIT** instruction (it does not contain an **AFTER** clause), the names of the jobs are copied to the work data as symbolic variables, initialized to '**SUBMITTED**'.
- If the **SUBMIT** is a conditional **SUBMIT** instruction (that is, it *does* contain an **AFTER** clause), then:
  * The **AFTER** is converted to a series of IF instructions similar to:

    IF %JOB1='ENDED' THEN SUBMIT ...

  * The names of the jobs are copied to the work data as symbolic variables initialized to " or null.
- The **SUBMIT** instruction then submits the job, but adds a **CHECKNET** instruction to the end of the job. **CHECKNET** uses the instructions in the work file as input to Jol. Jol reads the work file, alters the appropriate null symbolic variable to '**ENDED**', and rewrites the file. Then Jol reads the file again, and this time submits other jobs, if the required conditions are met.

Dependent Job Networks

This section of the manual discusses another method you can use to chain together many jobs to run in a particular sequence. The examples shown here demonstrate how you can **SUBMIT** a job from a currently executing job. The previous chapter showed how you can have Jol manage a network and **SUBMIT** a job or jobs after a particular job or group of parallel running jobs has ended.

The major differences between the two methods are:

Dependent job processing allows you to **SUBMIT** a job at any time from the current job. You code the **SUBMIT** instructions at appropriate points in your Jol program. The **SUBMIT** commands can be coded before the job ends, and allows for parallel processing, but you cannot wait for two jobs to terminate before submitting another.

With *Job Networking*, Jol manages the submission of jobs and controls *parallel* execution of jobs and the waiting for the termination of jobs before submitting other jobs in the network. Basically, you code your Jol programs as though they are independent jobs, and then code a special set of Jol network instructions that then submit the jobs in the desired sequence for you.

Jobs are usually submitted only when the current job ends, although you can force Jol to examine the status of the network at any time, and depending on the status, submit other jobs.

It is recommended that you be familiar with, or review, the **SUBMIT** and **SAVESYMS** commands. These commands, with their protocols, are explained in the *Jol Reference Manual* and *Jol Reference Guide*, and in the Appendices of this book.

Dependent Job Processing

Jol provides facilities to effectively schedule dependent jobstreams. That is, the dependent job is not compiled or placed on the system job queue until the current job has reached a point where it is desirable or safe to run the dependent job. By using this technique, a linkage can be created which controls the sequence of running a number of jobs.

Figures 3-1 and 3-2 on the following pages illustrate the use of these commands.

An Example of Dependent Job Processing.

Statement Number	Jol Code
JOB1: *etc.*	
.	
.	
Jol code	
.	
.	

```
 1      STEP1:          RUN PROGRAM1 ;
 2              IF STEP1=0 THEN
 3              SUBMIT JOB2 ;
 4      STEP2:          RUN PROGRAM2 ;
.
.
.
```

Explanation:

Statement 1 Instructs Jol to execute **PROGRAM1**.
Statement 2 Checks the completion code of **PROGRAM1**.
Statement 3 Instructs Jol to **SUBMIT JOB2** to the system job queue if the conditions are satisfied.
Statement 4 Continues the processing of **JOB1**.

Figure 11: Dependent Job Processing Example (1)

Passing Symbolic Variables and Submitting Other Jobs

Creating a more complex Dependent Jobstream for a particular application, which may consist of 2 or more jobs, requires certain considerations. You need to determine:

- The sequence of the run i.e., **JOB1**, **JOB2** etc.
- All of the variable symbolic names across the jobstream.
- The JOB to program dependencies.

Table 3-2 shows a simple example of a Payroll Application system.

Note: The following example shows one method that can be used to pass symbolic data from one job to another. Another method to pass symbolic variables from Job to Job is to use the SYMS Parameter with the SUBMIT Command. See the SUBMIT Command details for further information.

Sample Dependency Form

MIS System Dependency Chart Form Number MIS-DP1

System ID PAYROLL **Prepared by**: **Date:**

Job#	JOBNAME	DEPENDENCIES	Required Symbolic Values from System Scheduler
1	PAYJOB1	-----	%Cycle, %State, %Div
2	PAYJOB2	JOB1 After STEP20	%State, %Div
3	PAYJOB3	JOB1 After STEP25	%Cycle
4	PAYJOB4	JOB3	
5			
6			
7			
8			
9			
10			
11			

Figure 12: Example of a Payroll Jobstream

Note the following concerning the example above:

1. There are four jobs in the Payroll system.
2. Job **PAYJOB2** can be run after job **PAYJOB1** has completed **STEP20** successfully.
3. Job **PAYJOB3** can be run after job **PAYJOB1** has completed **STEP25** successfully.
4. Job **PAYJOB4** can be run after job **PAYJOB3** has completed successfully.
5. The symbolic variables **CYCLE**, **STATE**, **DIV** are used by the Payroll system.

Steps Required to Create the Jobstream

To create a sequence of jobs as described above, and to pass the required symbolic variable data through to each job automatically may be done in a manner similiar to that described below.

Step 1

Create a 'PANEL' in **PAYJOB1** to request the values for the variables (see table 12). Figure 13 below shows an example of a 3270 **PANEL** for the job.

```
PAYROLL SYSTEM ENTRY PANEL FOR JUNE


PLEASE ENTER THE CYCLE NUMBER          ===> _
PLEASE ENTER THE STATE CODE            ===>
PLEASE ENTER THE DIVISON CODE          ===>
```

Figure 13: Example of the Creation of a PANEL.

Step 2

Once the content of the supplied data has been verified, store the values of the variables in a member of a PDS for use by the other jobs, or in case of a rerun or restart. This process will eliminate the need to re-input the values. To store the data, use the **SAVESYMS** command. For example:

```
SAVESYMS CYCLE,STATE,DIV
        IN 'dataset-name(member)';
```

Step 3

In order for **PAYJOB1** to **SUBMIT PAYJOB2** and **PAYJOB3** at the desired points, code the command for Jol to **SUBMIT PAYJOB2** and **PAYJOB3** (see example below).

```
.
.
.
Jol code
.
.
.
 STEP20:      RUN PROG20 ;          /* Jol Instruction to
                                       Execute PROG20*/
              IF STEP20=0 THEN      /* Check the Condition
                                       Code*/
              SUBMIT PAYJOB2;       /* If satisfied,
                                       SUBMIT PAYJOB2*/
.
.
.
Further Run Instructions
.
.
.
```

```
STEP25:     RUN PROG25 ;
            IF STEP25=0 THEN
            SUBMIT PAYJOB3 ;
```

Figure 14: Dependent job Processing Example (2)

When Jol activates the **SUBMIT** command, the job is placed on the system job queue and runs as an independent job.

Step 4

When creating the Jol code for **PAYJOB2** and **PAYJOB4** - both require symbolic substitution - issue, as part of the code, an instruction for Jol to **INCLUDE** the stored symbolic values from **PAYJOB1**. The format of the instruction is:

INCLUDE member-name ;

Now the need for entering data values again is eliminated, which reduces the incidence of error.

Step 5-

For **PAYJOB3** to **SUBMIT PAYJOB4**, use the method outlined in Step 3.

Note the following concerning this example:

The system outlined in the example is very simple, however, if required, far more complex schedules can be handled with ease. Using Jol's **CALENDAR**, dependencies based on the day of the week (e.g. Friday), month, year, and so on, can be incorporated into the code so that correct schedules are run automatically without constant supervision or manual decision.

Jobs are not placed on the system jobqueue, to be released in error, unless they are able to run. This removes the need for Operator decision because **"IF ITS ON THE QUEUE RUN IT"** becomes a true statement.

Scheduling Jobs by Date

Scheduling

Jol's Scheduling System is extremely simple to use and written *entirely in high level Jol Code*; this allows you to easily make changes to the system to suit your installation. Using the Scheduling Facility, it is possible to program job schedules for days, months, or even years in advance. All your existing jobs can be started without changes. Furthermore, Jol can be used to schedule your de-bugged and already tested jobs written in **JCL**. Full Jol symbolic variable processing is allowed, even with JCL jobs.

The Jol Schedule Data Set contains various members that can be used to specify which jobs are to run on **WORKDAYS, HOLIDAYS, MONDAYS, TUESDAYS**, etc., and jobs for particular dates such as **JUL12**. At a time suitable to you, Jol examines this data set and prepares all the appropriate jobs for execution.

> Note: So that you can submit jobs on the day(s) before or after a Holiday, there is prototype code provided in member **SPECIAL** of the Schedule Data Set.

In general, to specify which jobs are to be started, code the name of the member containing the job in the **JOL.SCHEDULE** data set using the **PREPARE** command. For example, to indicate which jobs are to be executed daily is done by coding the names of the jobs in member **WORKDAY**, as shown below.

```
PREPARE BACKUPS;
PREPARE ACCOUNTS;
PREPARE PAYROLL;
```

Figure 15: Indicating Which Jobs are to Run.

The jobs named are submitted for execution on the appropriate days. Jobs may contain any Jol instruction, and include networking instructions. For example, even daily jobs can examine the Jol calender or accept information from the terminal and dynamically alter themselves before submission.

Rules

- You may code more than one job name per line.
- The jobs named are compiled by Jol, and then submitted for execution. Jobs may contain any Jol instruction, including Jol Job Networking instructions. For example, some jobs, even if submitted daily, may examine the Jol calender or accept information from the terminal using the Jol **PANEL** instruction and dynamically alter themselves before submission.
- Even if a job is mentioned twice or more frequently, it will be submitted once only. For example, if a job is to be submitted on the first working day of January, but is also to be submitted on **MONDAY**, and the two coincide, only one job will be submitted.

Planning Ahead

Jobs in member **WORKDAY** are submitted on a daily basis. In addition, other members of the library are examined based on dates. The following members of the schedule data set are executed on the appropriate days:

WORKDAY	Executed only on normal working days.
SPECIAL	Executed every day, *see later pages for details*
HOLIDAY	Executed only on non-working days.
WEEKEND	Executed only on Weekends.
SUN	Executed on **SUNDAY**s.
MON	Executed on **MONDAY**s.
TUE	Executed on **TUESDAY**s.
WED	Executed on **WEDNESDAY**s.
THU	Executed on **THURSDAY**s.
FRI	Executed on **FRIDAY**s.
SAT	Executed on **SATURDAY**s.

Figure 16: Daily Members Examined by the Schedule Facility.

In addition, the following members of the schedule data set are executed *(if present)* on the specified days:

JAN01	Executed on January 1st.
JAN02	Executed on January 2nd.
...	
FEB01	Executed on February 1st.
MAR01	Executed on March 1st.
APR01	Executed on April 1st.
MAY01	Executed on May 1st.
JUN01	Executed on June 1st.
JUL01	Executed on July 1st.
AUG01	Executed on August 1st.
SEP01	Executed on September 1st.

OCT01	Executed on October 1st.
NOV01	Executed on November 1st.
DEC01	Executed on December 1st.

Other jobs can be submitted on the last working day, the last day, the last working day -1 and so on. A member called **SPECIAL** has prototype code for these types of jobs.

Figure 17: Month and Day Members Containing Scheduling Information.

MONWK01	Executed on the *first* Monday of the year.
MONWK02	Executed on the *second* Monday of the year.
MONWK*nn*	Executed on the *nth* Monday of the year.
...	
TUEWK01	xecuted on the *first* Tuesday of the year.
TUEWK*nn*	Executed on the *nth* Tuesday of the year.
...	
WEDWK01	Executed on the *first* Tuesday of the year.
THUWK01	Executed on the *first* Tuesday of the year.
FRIWK01	Executed on the *first* Tuesday of the year.
SATWK01	Executed on the *first* Tuesday of the year.
SUNWK01	xecuted on the *first* Tuesday of the year.

Using **MONWK01** etc allows you to submit jobs every second week, should you so desire.

Figure 18: Week Number and Day Members to be Examined

How the Schedule Facility Starts the Required Jobs

Once a day, Jol examines the Schedule data set. If it is not a holiday, it examines the **WORKDAY** member; if it is Saturday or Sunday, the data in these members is used instead, otherwise member **HOLIDAY** is used. Additionally, any other member such as **JUL14** is also examined, if appropriate.

- The names of the jobs found are converted to Jol **INCLUDE** statements and copied to a working file, each separated by an *** JOL;** separator card.
- Jol then reads the file, and executes any instructions found therein as usual.
- The jobs are then submitted.

Below is a sample of the **WORDAY** member of the Schedule Data Set.

```
      /* This is the normal WORKDAY member of the JOL.SCHEDULE Data Set.

   It is examined on WORKING DAYS ONLY.

   You may put in other Jol commands before you use the
   PREPARE verb to prepare your work for submission.
   See the example below for 'SPEC'.

*/

 Prepare DAILYBU;          /* Prepare Daily Backup Job */
 PREPARE INVOICES;         /* Prepare Daily INVOICE Job */

 if %day='TUESDAY'         /* If today is TUESDAY */
 then prepare  SPEC;       /* then prepare a SPEC job */
```

Figure 19: Sample WEEKDAY Jobs

Catering for Special Requirements not Handled by Schedule

The ability to automatically submit jobs on every **WORKDAY**, **MONDAY** or even on **ODD** or **EVEN** weeks etc is highly flexible. However, sometimes you also need to be able to submit work on the day before a **HOLIDAY**, the day after a **WORKDAY** and so on. One way to do this is to place your **PREPARE** commands in the appropriate position in the **SPECIAL** member of the Jol Schedule file.

Before the **SPECIAL** file is examined, an array is set up so that you can easily determine if today, tomorrow, or today plus three days is a Holiday, or a Work Day. In addition to all the usual Jol Calendar Variables such as **%DAY, %DAYNO, %MONTH** etc, another variable called **%LASTDAY** contains the number of days in the current month. **%LASTDAY** is correct for Leap Years.

Using Jol programming language and the variables above makes special coding requirements easy to cater for.

The following page shows some of the coding that you may do in the **SPECIAL** file.

WORKDAY Group

Tomorrow is a Holiday

The day after tomorrow is a Holiday

The day after that is a Holiday

Yesterday was a Holiday

Another Method for Testing if Yesterday was a Holiday

Tomorrow is the last WORKDAY in the week

The day after tomorrow is the last WORKDAY

The day after that is the last WORKDAY

```
 /* This is the SPECIAL member of the JOL.SCHEDULE Data Set.            00010000
                                                                       00020000
    It is examined every day, and can test if TOMORROW is a            00030000
    Holiday, or yesterday and so on.  See the examples below. */       00040000
                                                                       00050000
    if %holiday='YES'                                                  00060000
    then do;                                                           00070000
         write 'We could PREPARE a job here';                          00080000
    end;                                                               00090000
                                                                       00100000
/* See if the next few days are HOLIDAYS */                           00110000
                                                                       00120000
    if %holiday='NO'                                                   00130000
    then do;                                                           00140000
         if %list(%dayno+1)='HOL'                                      00150000
         then do;                       /* TOMORROW is a Holiday */    00160000
             write 'We could PREPARE a job here';                      00170000
             PREPARE tomorrow;                                         00180000
             PREPARE B ;                                               00190000
         end;                                                          00200000
         if %list(%dayno+2)='HOL'                                      00210000
         then do;                       /* The NEXT day is a Holiday */00220000
             write 'We could PREPARE a job here';                      00230000
         end;                                                          00240000
         if %list(%dayno+3)='HOL'                                      00250000
         then do;                       /* and +3 day is a Holiday */  00260000
             write 'We could PREPARE a job here';                      00270000
         end;                                                          00280000
         if %list(%dayno-1)='HOL'                                      00290000
         then do;                       /* YESTERDAY was a Holiday */  00300000
             write 'We could PREPARE a job here';                      00310000
         end;                                                          00320000
    end;                                                               00330000
                                                                       00340000
/* Sample ONLY:- See if Yesterday was a Holiday, and Today is a       00350000
   Workday.                                                        */  00360000
                                                                       00370000
    if %list(%dayno-1)='HOL'            /* Was YESTERDAY a Holiday */  00380000
    &  %list(%dayno)='WORK'             /* and TODAY a WORKDAY ?   */  00390000
    then do;                                                           00400000
        write 'We could PREPARE a job here for Yesterday''s Holiday';  00410000
        PREPARE yesthol;                                               00420000
        PREPARE B ;                                                    00430000
    end;                                                               00440000
                                                                       00450000
/* See if the next few days are the END OF MONTH */                   00460000
                                                                       00470000
    if %holiday='NO'                                                   00480000
    then do;                                                           00490000
         %work=%dayno+1;                /* Set WORK to TOMORROW  */    00500000
         if %work=%lastday then do;     /* TOMORROW is the LAST  */    00510000
                                           DAY of the month      */    00520000
             write 'We could PREPARE a job here';                      00530000
             PREPARE B ;                                               00540000
         end;                                                          00550000
         %work=%work+1;                 /* Check the NEXT Day    */    00560000
         if %work=%lastday then do;     /* The NEXT day is the   */    00570000
                                           LAST DAY of the month */    00580000
             write 'We could PREPARE a job here';                      00590000
             PREPARE B ;                                               00600000
         end;                                                          00610000
         %work=%work+1;                 /* Check the Day AFTER that */ 00620000
         if %work=%lastday then do;     /* The NEXT day(+3) is the */  00630000
                                           LAST DAY of the month */    00640000
             write 'We could PREPARE a job here';                      00650000
             PREPARE B ;                                               00660000
         end;                                                          00670000
    end;                                                               00680000
```

Figure 20: Sample SPECIAL File.

Telling Jol Which Days are HOLIDAYS

The Scheduling System uses Jol's automatic Calendar facility to determine if the day of the week is a Saturday or Sunday, and automatically submits any jobs for the appropriate day.

However, Jol does not know which days are holidays in your country for any given year, or which other days in your organization are holidays for special reasons such as a nine day fortnight instead of the more usual ten day fortnight.

So that the Jol Scheduling System can determine which days are Holidays, Rostered Days Off and so on, you code these special days in a member called **HOLS1988, HOLS1989** etc. The format is as the example for **HOLS1987** shown below:

```
JAN 1        /* AUSTRALIAN HOLIDAYS   */      00010000
JAN 1        /* NEW YEAR'S DAY        */      00020000
JAN 26       /* AUSTRALIA DAY         */      00030000
APR 17       /* GOOD FRIDAY           */      00040000
APR 18       /* EASTER SATURDAY       */      00050000
APR 19       /* EASTER SUNDAY         */      00060000
APR 20       /* EASTER MONDAY         */      00070000
APR 25       /* ANZAC DAY             */      00080000
JUN 8        /* QUEENS BIRTHDAY       */      00090000
DEC 25       /* CHRISTMAS DAY         */      00100000
DEC 26       /* BOXING DAY            */      00110000
```

Figure 21: Example of the HOLIDAY member in the Schedule Data Set.

As you can see in the example above, you code the the first three characters of the month, followed by at least one space and any optional comments. Note that each day must be on a separate line.

Because Jol uses the year as part of the member name, you can set up holidays for years in advance.

Other Useful Facilities for Use in Scheduling

Additionally, Jol allows:

- Read and Write access to existing or new data sets at compile time.

  You may have data sets that contain other scheduling information or further details of work to be run on particular days, or under particular circumstances, or both. Using this data, Jol can create tailored job streams. These job streams can be created on finding the appropriate data in the schedule.

- Access to the system calendar. Jol allows the date (year, month, day) and time to be accessed. Using this data, Jol can create tailored job streams.

- Testing for the existence of data sets. Jol can test if a data set exists. If a data set does exist, it is possible to submit other jobs to run, or take a different path through the current job.

- Symbolic Parameters or Variables can be passed from Job to Job.

Implementation

Language

So that the Jol Networking and Scheduling system can be made available on a wide variety of computers, the system is implemented using the tools available in Jol. Because it is written in high level Jol, should your installation desire to change any part of the facility, the Jol code may be changed with the standard editing facility.

Data Sets Required *for NETWORKING*

For mainframe computers, three Partitioned Sets must be allocated before use: The MS/DOS version allocates and frees the data sets as necessary. Two data sets are used by the Networking Facility. These are:-

1. **SYS2.JOLNET.STATUS** Contains the current status of the Network and can be viewed using SPF to determine the status of any network. Members names are the same as the name of the network coded on the NETWORK command.

   Each member contains symbolic variables as shown below.

2. **SYS2.JOLNET.CODE** Contains the **IF** and **SUBMIT** statements necessary to drive the network.

   Members names are the same as the name of the network coded on the NETWORK command.

You should allocate these data sets with the same DCB information as your primary Jol **INCLUDE** and **CMDLIB** data sets, and they must be large enough to run all the jobs throughout the day as data is constantly written to them when jobs are submitted. If the data sets fill, the jobs being networked must be restarted. You can temporarily delay Networked jobs to compress the files should you wish.

As a starting point, it is suggested that you allocate 5 cylinders for the **STATUS** data set, and 2 cylinders for the **CODE** data set. You can examine them at the end of the day, and free any unused storage.

Data Sets Required for SCHEDULING

When Jol is installed, a data set called **SYS2.JOL.SCHEDULE** is loaded on to your system. Depending on the number of jobs you have in your daily schedule, you may have to reallocate it with more storage. The Scheduling System uses **SYS2.JOL.SCHEDULE** for INPUT only when you run the **SCHEDULE** Clist, and therefore the data set will fill only when you edit it with ISPF or PFD; should it become filled, use your standard recovery methods.

Method of Operation

To use the Network Facility, five Jol commands are required. These are:

NETSUB
SUBMIT
NETWORK
CHECKNET
ENDNET

The following process occurs when the macros are executed:

The NETWORK Command

The **NETWORK** instruction first checks the variable **%SYSNET**. If **%SYSNET** is not blank, then a NETWORK instruction has been executed without an **ENDNET** instruction, and an error message is issued.

Otherwise, it sets **%SYSNET** to the name of the network and opens the **STATUS** and **CODE** data sets, using the name of the network as a member name. The **SUBMIT** command will copy information into these members. Both data sets are used at execution time, and checked at the end of every job in the network.

It also sets an indicator that is used by the **SUBMIT** command to determine if the **SUBMIT** is a networked, or non-networked command.

The SUBMIT Command

When a **SUBMIT** command is executed, it does the following:

- If the indicator mentioned above is not set, the **SUBMIT** instruction operates as usual. That is, the job is submitted to the operating system for execution.

  Otherwise, the **SUBMIT** takes special actions for controlling the network.

- If the **SUBMIT** is an unconditional **SUBMIT** instruction (it does not contain an **AFTER** clause), the names of the jobs are copied to the work data as symbolic variables, initialized to '**SUBMITTED**'.

- If the **SUBMIT** is a conditional **SUBMIT** instruction (that is, it *does* contain an **AFTER** clause), then:

  * The **AFTER** is converted to a series of IF instructions similar to:

    IF %JOB1='ENDED' THEN SUBMIT ...

  * The names of the jobs in the network are copied to the **STATUS** data set as symbolic variables initialized to '' or null.

- The **SUBMIT** instruction then creates a **NETSUB** command, and copies it to the **CODE** data set.

The ENDNET Command

The **ENDNET** instruction performs various checks, sets **%SYSNET** to null, places a **CHECKNET** instruction in the **CODE** data set, closes the **STATUS** and **CODE** data sets, and exits.

The CHECKNET Command

CHECKNET runs at execution time. It uses the instructions and data in the **CODE** and **STATUS** data sets work file as input to Jol. Jol reads the work file, alters the appropriate null symbolic variable to '**ENDED**', and rewrites the file. Then Jol reads the file again, and this time submits other jobs, if the required conditions are met.

The NETSUB Command

NETSUB also runs at execution time. It submits any required jobs for execution. It also adds **CHECKNET** instructions to any jobs it submits. Therefore when any job in the network terminates, it issues a **CHECKNET** instruction, and the **CHECKNET** determines if any other jobs in the network can be submitted.

Documented

The following example shows how the Networking Facility operates. In this example, **JOB1** and **JOB2** may proceed in parallel. The other jobs must execute

Example

in the following sequence:

- **JOB3** can commence processing after *either* **JOB1** or **JOB2** have ended.
- **JOB4** cannot commence until *both* **JOB1** and **JOB2** have finished.
- **JOB5** must wait for **JOB2** to end before it can start.
- **JOB6** must wait for **JOB3, JOB4 and JOB6** before it can start.
- **JOB7** must wait for **JOB6** to end, and it is to be executed *only on Fridays*.

If the job is coded as:

```
NETWORK TWO;
     SUBMIT JOB1;
     SUBMIT JOB2;
     SUBMIT JOB3
               AFTER JOB1 OR JOB2 ENDED;
     SUBMIT JOB4
               AFTER JOB1 AND JOB2 ENDED;
     SUBMIT JOB5
               AFTER JOB2 ENDED;
     SUBMIT JOB6
               AFTER JOB3 & JOB4 & JOB5 ENDED;
     IF %DAY='FRIDAY' THEN
          SUBMIT JOB7
               AFTER JOB6 ENDED;
ENDNET;
```

then after the Jol Compiler has read these statements, it submits the first job which effectively contains only a **CHECKNET** instruction. This first **CHECKNET** instruction starts the network by submitting the first two jobs. In addition, the **CODE** and **STATUS** data sets will be set up as follows:

In member **TWO** of the **SYS2.JOLNET.STATUS** data set will be the following:-

```
%JOB1='';
%JOB2='';
%JOB3='';
%JOB4='';
%JOB5='';
%JOB6='';
%JOB7='';
```

and in member **TWO** of the **SYS2.JOLNET.CODE** data set will be the following:-

```
INCLUDE TWO;      /* Read the status of the Job names */
```

```
IF %JOB1='ENDED'
&  %JOB2='ENDED'
THEN NETSUB JOB3  syms etc;

IF %JOB1='ENDED'
|  %JOB2='ENDED'
THEN NETSUB JOB4  syms etc;

IF %JOB2='ENDED'
THEN NETSUB JOB5  syms etc;

IF %JOB3='ENDED'
&  %JOB4='ENDED'
&  %JOB5='ENDED'
THEN NETSUB JOB3  syms etc;

IF %JOB6='ENDED'
THEN SUBMIT JOB7  syms etc;

SAVESYMS JOB1,JOB2, JOB3, JOB4, JOB5, JOB6, JOB7
      IN 'SYS2.JOLNET.STATUS(TWO)';
```

After **JOB1** has run, the **CHECKNET** command is executed. It reads the member **TWO** of the **SYS2.JOLNET.STATUS** data set, and alters it as shown below:-

```
%JOB1='ENDED'
%JOB2='SUBMITTED';
%JOB3='';
%JOB4='';
%JOB5='';
%JOB6='';
%JOB7='';
```

The Jol Compiler now reads the *updated* members, and processes the instructions. In this case, no other jobs will be submitted.

After **JOB2** has run, the **CHECKNET** command is again executed. It reads the member **TWO** of the status data set, and alters it as shown below:-

```
%JOB1='ENDED'
%JOB2='ENDED';
%JOB3='';
%JOB4='';
%JOB5='';
%JOB6='';
%JOB7='';
```

The Jol Compiler now reads the *updated* member, and processes the instructions.

In this case, other jobs will be submitted.

This process continues until all the jobs have been run.

Jol Scheduling and Job Networking Facilities

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,
P O BOX 417
Ballan,
Victoria
AUSTRALIA, 3342

Telephone + 61 (0)401 054 155
Email clementclarke@ozemail.com.au.