SPANEX™

SPANEX Scheduling Beginning User's Guide

Span Software Consultants Limited

Version: 06.0 Product Number: SPOS-001

Revision: 1st May 2015 Manual Ref: SPX-12-010

CONTENTS Page

DIAGRAMS Page

FIGURES Page

1 Introduction

This manual is intended for new users of the SPANEX™ Job Scheduling facilities. It is written in tutorial style, and takes the reader through the basics of defining simple job dependencies up to a relatively complex application with calendar definitions and interactions with other systems.

If desired, the examples given can be worked on a real system, with simplified "dummy" jobs being used instead of full application programs. Suitable JCL for such dummy jobs is given in Figure 5 on page 11 of this manual. Jobs used in the examples are given arbitrary names of the form "JOBnnn"; these job names should be replaced with names that conform to your installation standards, but in a consistent way. For example, all occurrences of JOB100 should be replaced by the same standard jobname. If this is not done carefully, unexpected or erroneous results may occur.

It is assumed that the SPANEX software has been installed on the system, and that such items as a JCL procedure for RCM generation have been created. Instructions for this are given in other SPANEX manuals, including the SPANEX Installation and Maintenance Manual, and the SPANEX Restart and Job Networking Guide. Consult your SPANEX systems programmer to determine any special aspects of SPANEX in your own installation.

For the purposes of the examples in this manual, it is assumed that an MVS system with TSO is being used. SPANEX will also run with a dedicated 3270 terminal instead of TSO. Consult your SPANEX systems programmer if you do not have an MVS system with TSO available.

SPANEX Manuals	Order No
SPANEX General Usage Manual	SPX-02
SPANEX Restart and Job Networking Guide	SPX-03
SPANEX Scheduling Beginning User's Guide	SPX-12
SPANEX Automated Data Areas Manual	SPX-04
SPANEX Messages and Codes Manual	SPX-05
SPANEX Terminal User's Guide	SPX-07
SPANEX Installation and Maintenance Manual	SPX-09
SPANEX Documentation Index	SPX-10
Span Macros Manual	SPZ-02
Span Service Routines Manual	SPZ-03
SPSMFINF User Manual	SPI-01

2 Starting with SPANEX Job Scheduling

2.1 A Simple Job Sequence

The dummy application described in this manual is developed as the chapters progress. It starts with the following simple sequence of jobs:

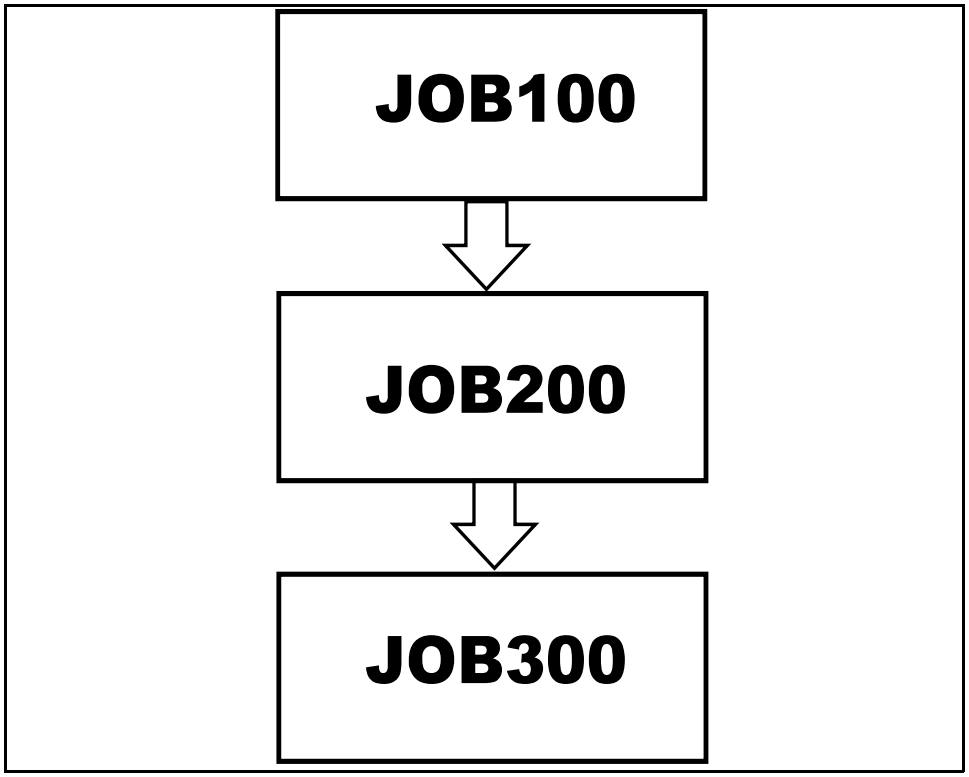

Diagram 2. A Simple Job Sequence

These three jobs, named JOB100, JOB200 and JOB300, together constitute an application system. The three jobs must execute in sequence in order to perform the application processing. Naturally, each job must complete successfully before the following job can be allowed to execute.

In SPANEX terminology, a related group of jobs, such as this, is known as a Job Network. As we shall see later, SPANEX does not impose any software limitations on the number of jobs in a Network, or on the number or complexity of relationships between them.

2.2 Defining the Jobs to SPANEX

The jobs in a Job Network are defined to SPANEX in a control block known as a Restart Control Module or RCM. The reason for this name is that the RCM also contains information relating to the use of the SPANEX Automatic Job Restart facility.

The RCM is defined by the use of SPANEX macro statements, which are then compiled using the RCMGEN procedure, which is described in the SPANEX Restart and Job Networking Guide. This procedure places the compiled RCM in an RCM Library, which is a standard operating system load library.

For the purposes of this learning exercise, we will create a temporary RCM library to contain the RCMs generated for the dummy application we are defining.

Run a job similar to the one shown below to allocate space for the RCM library. Note that you may have to modify the specification of UNIT and VOLUME information for the device on which the library is to be created, and that the dataset name may need to be modified to conform to your installation standards.

```
//jobname  JOB  (accounting info),'SPANEX', . . .
//*  ALLOCATE SPANEX RCM LIBRARY
//         EXEC PGM=IEFBR14
//RCMLIB   DD   DSN=SPANEX.TEST.RCMLIB,
//         UNIT=SYSDA,DISP=(,CATLG),
//         DCB=(RECFM=U,BLKSIZE=6233),
//         SPACE=(CYL,(2,1,10))
//
```

Figure 1. Allocating an RCM Library

Initially, we will define our application using the SPANEX QUICKNET facility, which is the simplest way of defining SPANEX Job Scheduling. Using QUICKNET does not limit the use of the many Scheduling facilities within SPANEX, but it permits only limited exploitation of SPANEX's Automatic Job Restart capabilities, and it imposes some restrictions on the techniques for submitting jobs to the operating system. However, these limitations need not concern us at this point.

The statements required to generate our initial RCM are shown below:

```
JOB100   QUICKJOB
JOB200   QUICKJOB  PREREQ=JOB100
JOB300   QUICKJOB  PREREQ=JOB200
RCM100   QUICKNET  LIBRARY=PDS
```

Figure 2. Initial RCM definition statements

Note the following points about this sequence of statements:

a) Each job in our network is represented by a QUICKJOB statement.

b) Where there is a dependency on another job, this is represented by means of the PREREQ= parameter. On a QUICKJOB statement, the PREREQ= parameter lists one or more jobs that must have completed successfully before this job can start.

c) The sequence of statements is terminated by a QUICKNET statement. The QUICKNET statement defines any options that we wish to specify for the Job Network as a whole. In order for SPANEX to schedule jobs, it needs access to our JCL. For use of the QUICKNET facility, this JCL must reside on a JCL library. In our example, the JCL library is to be a partitioned dataset.

d) Note that we have given the dummy application a SPANEX Network name of RCM100. This is an arbitrary name. When SPANEX is in production use, you will probably develop a naming convention for RCMs, or else use meaningful mnemonic names.

Now we are ready to generate our initial RCM generation. Run a job similar to that shown below, in order to achieve this:

```
//jobname  JOB  (accounting info),'SPANEX', . . .
//*  GENERATE SPANEX RCM FOR DUMMY APPLICATION
//         EXEC RCMGEN,RCM=RCM100
//RCMASM.SYSIN  DD  *
JOB100   QUICKJOB
JOB200   QUICKJOB  PREREQ=JOB100
JOB300   QUICKJOB  PREREQ=JOB200
RCM100   QUICKNET  LIBRARY=PDS
//RCMLKED.SYSLMOD  DD  DSN=SPANEX.TEST.RCMLIB(RCM100),
//       DISP=SHR
```

Figure 3. Generating the initial RCM

Depending on your SPANEX installation options, it is possible that you may receive an error in executing the RCM generation job, accompanied by a message in the output from the job such as:

```
SPXDEF25  ERROR - GLOBAL LOG NOT DEFINED
```

In this case, please refer to section 4.1 on page 21 of this manual for an explanation of the SPANEX Global Log facility. You will need to add the GLOGDD= and GLOGDSN= parameters to your QUICKNET statement.

2.3 Setting up the Application JCL

Now we will create our JCL Library, and set up some JCL for the three jobs in our application. We will also take this opportunity to allocate a SPANEX Command Library dataset, which we will use later in Chapter 5. Run two jobs similar to those shown below, in order to accomplish this. Note that UNIT, VOLUME and dataset name information may need to be changed to conform to your installation standards.

```
//jobname  JOB  (accounting info),'SPANEX', . . .
//*  ALLOCATE SPANEX JCL LIBRARY
//         EXEC PGM=IEFBR14
//JCLLIB   DD   DSN=SPANEX.TEST.JCLLIB,
//         UNIT=SYSDA,DISP=(,CATLG),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),
//         SPACE=(CYL,(2,1,10))
//UTILLIB  DD   DSN=SPANEX.TEST.UTILLIB,
//         UNIT=SYSDA,DISP=(,CATLG),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),
//         SPACE=(CYL,(2,1,10))
//
```

Figure 4. Allocating a JCL Library

```
//jobname  JOB  (accounting info),'SPANEX', . . .
//*  ADD APPLICATION JCL TO SPANEX JCL LIBRARY
//         EXEC PGM=IEBUPDTE,PARM=NEW
//SYSUT2   DD   DSN=SPANEX.TEST.JCLLIB,DISP=OLD
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DATA
./ ADD NAME=JOB100
//JOB100   JOB  (accounting info),'SPANEX', . . .
//STEP1    EXEC PGM=IEFBR14
//STEP2    EXEC PGM=IEFBR14
//
./ ADD NAME=JOB200
//JOB200   JOB  (accounting info),'SPANEX', . . .
//STEP1    EXEC PGM=IEFBR14
//STEP2    EXEC PGM=IEFBR14
//
./ ADD NAME=JOB300
//JOB300   JOB  (accounting info),'SPANEX', . . .
//STEP1    EXEC PGM=IEFBR14
//STEP2    EXEC PGM=IEFBR14
/*
```

Figure 5. Adding JCL to the JCL Library

Note that it is not a SPANEX requirement that the job name is the same as the PDS member name, although, for simplicity, we are using the job name as a member name in each case for our examples. Of course, if you

prefer, you can create members of the JCL library by means of an online editor, such as TSO/ISPF.

2.4 Preparing to use Full-Screen TSO

In order to control SPANEX, you use the SPANEX Utility commands. The SPANEX Utility can execute in batch mode, from an operator console, or from TSO. During this exercise, you will use the full-screen TSO option. This assumes you have a 3270-type terminal from which you can access TSO. If this is not the case, consult your SPANEX systems programmer.

Write a TSO CLIST similar to the one shown below, and add it to your CLIST library, with a member name of SPX. If you are unsure how to do this, consult your SPANEX systems programmer.

```
PROC 1 RCMNAME
/*  Invoke SPANEX full-screen TSO support  */
/*  (foreground only)                      */
ALLOC F(JOBPDS)  DA('SPANEX.TEST.JCLLIB') SHR REUS
ALLOC F(SPXUTLIB)  DA('SPANEX.TEST.UTILLIB') SHR REUS
ALLOC F(TASKLIB) DA('SPANEX.TEST.RCMLIB') SHR REUS
SP SPXFTSO0 'NET=&RCMNAME' OPT(A)
END
```

Figure 6. A CLIST for SPANEX Full-Screen TSO support

Note that this CLIST has the effect of allocating your RCM Library to DDname TASKLIB, and your JCL Library to DDname JOBPDS. These are pre-defined SPANEX DDnames.

2.5 Running the Application

Now you are ready to run the application for the first time. To do this, you will use the full-screen TSO foreground interface to the SPANEX Utility.

Run the CLIST created in section 2.4 above by entering the TSO command:

```
%SPX RCM100
```

This command can be entered from the TSO "READY" prompt, or from the TSO command facility of ISPF.

Note that the name of the RCM is entered as a parameter of the command. The "%" at the beginning of the command is TSO shorthand notation to reduce the program search time - it informs TSO that the "SPX" command is a CLIST and not a program.

If you are presented with a row of three asterisks, press ENTER.

You will now see the SPANEX full-screen interface. Note the screen layout. The large area at the top of the screen is the area SPANEX uses to display messages and command responses to you. At the bottom of the screen is the command input area, and a password input area. We are not using Network passwords yet. Note that SPANEX has set the default Job Network name to "RCM100", your test RCM name. This is because of the parameter you entered to the CLIST.

To start a SPANEX Job Network executing, you use the NETSTART command. NETSTART tells SPANEX to prepare the Network for execution, and usually submits one or more jobs for execution to the operating system. The NETSTART command has a number of optional parameters, but for the first run, you will not need any of them.

In the command input area of your screen, type the command:

```
NETSTART
```

and press ENTER. SPANEX will respond with a message:

```
SPX857A  CONFIRM NETSTART FOR RCM=RCM100.  REPLY YES
OR NO
```

Because NETSTART is a significant process, SPANEX requests permission before executing it. We will see later how to avoid this additional prompt if you don't require it. Enter the word:

```
YES
```

in the command input area and press ENTER.

SPANEX will now submit JOB100, the first job in our simple network. The entire network will execute without further human intervention. If you are not quick with the next actions, you may miss the network execution altogether!

Press Program Function Key 2 on your keyboard to invoke the SPANEX "MAP" command facility. This gives a real-time display of the status of each job in a SPANEX Job Network. If your network is still running, you will see how it is progressing. If you see JOB100 displayed with a "status" value of PREX, it is possible that you have set up your job JCL incorrectly, or else that there is a large queue of jobs waiting to process. Check the job class you have specified in your JCL. If you cannot understand why the jobs are not running, consult your SPANEX systems programmer.

This page intentionally left blank.

3 A More Realistic Application System

3.1 Jobs and Jobsteps

With the advent of an automation system such as SPANEX, it becomes feasible for application systems to contain far more OS Jobs than in a manually-scheduled system.

The number of jobs is no longer limited by the fact that each job has to be manually submitted, and manually checked for successful completion.

In the past, jobs have tended to contain many Jobsteps, such that running the jobs was simple, but each job was itself very complex. JCL for such jobs is difficult to understand and to maintain, and tends to contain many OS Condition Code checking parameters, and many "passed" and temporary datasets.

A far better JCL architecture, and one that becomes possible with the use of SPANEX, is to make each OS job a simple one-step process. In reality, there may be more than one actual jobstep in each job, but a simple rule is to limit each job to a single application process, perhaps one application program, or one utility such as a sort execution. Within the same job there may sometimes be additional jobsteps, for example to allocate disk space, or to delete unwanted work datasets.

When the tasks of an application system are divided up in this way, the dependencies between the resulting jobs may look much more like that shown in Figure 6 on page 16 than the simple example we have already seen.

In an MVS system, this results in many more jobs running overall, and this allows MVS to use its powerful performance and load-balancing facilities to extract the maximum throughput from the hardware available. This is particularly true of multi-processor CPUs, which are becoming increasingly common.

We will now see how to implement the scheduling of this more complex job sequence using SPANEX. We will again use dummy jobs in order to illustrate the automatic scheduling process, but the actual function of each job is not important to the understanding of the use of SPANEX.

A further discussion of the effects of SPANEX on JCL, and the opportunities for improved JCL coding will be found in Section 6 on page 31 of this manual.

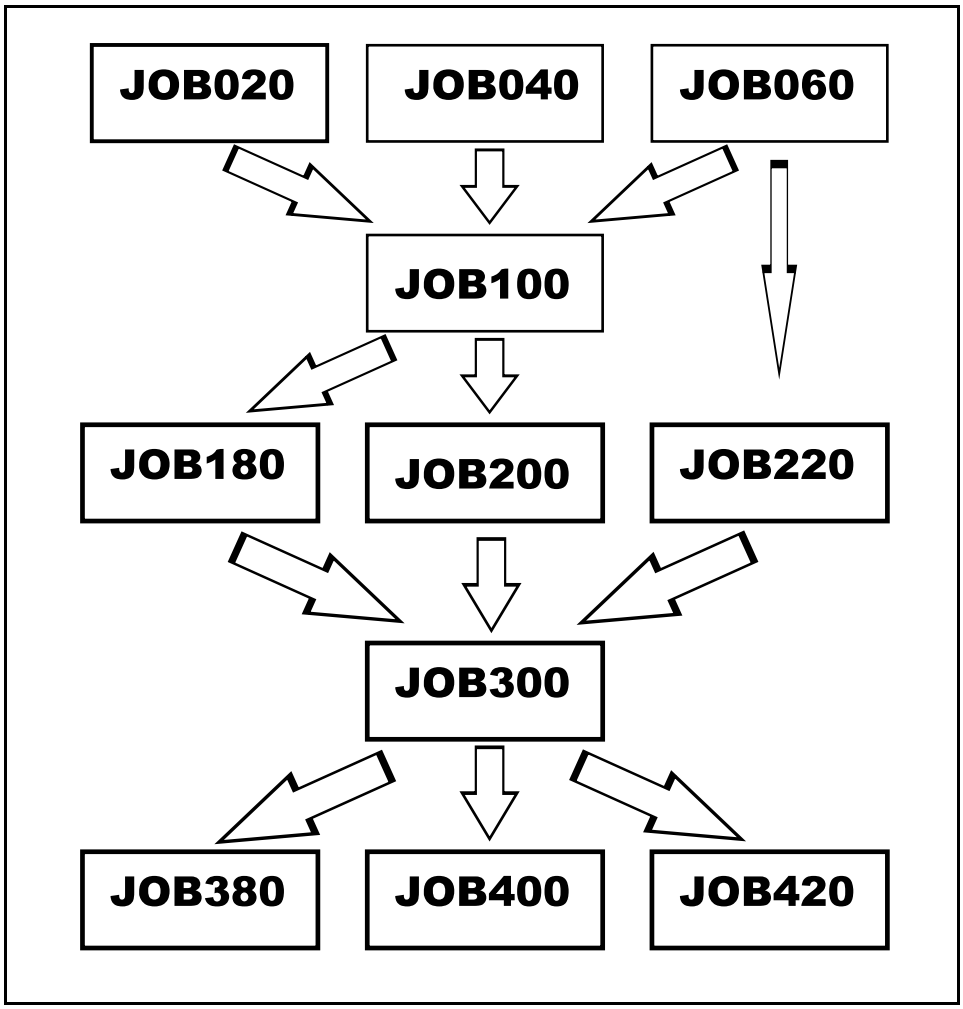

Diagram 3. A More Complex Job Network

SPANEX does not impose any limitations on the number of relationships between jobs, and any number of parallel streams of jobs can be defined, as allowed by the processing of the particular application system. In practice, when the jobs are running, the parallelism is limited by the number of available initiators in the system, although these may be on different machines in a multi-CPU JES2 shared spool, JES3 or SYSPLEX configuration.

The RCM definition statements for this more complex example are shown in Figure 7 below.

```
JOB020   QUICKJOB
JOB040   QUICKJOB
JOB060   QUICKJOB
JOB100   QUICKJOB  PREREQ=(JOB020,JOB040)
JOB180   QUICKJOB  PREREQ=JOB100
JOB200   QUICKJOB  PREREQ=JOB100
JOB220   QUICKJOB  PREREQ=JOB060
JOB300   QUICKJOB  PREREQ=(JOB180,JOB200,JOB220)
JOB380   QUICKJOB  PREREQ=JOB300
JOB400   QUICKJOB  PREREQ=JOB300
JOB420   QUICKJOB  PREREQ=JOB300
RCM200   QUICKNET  LIBRARY=PDS
```

Figure 7. RCM definition statements for a more complex example

Note that it is possible to define more than one pre-requisite job in the PREREQ= parameter of the QUICKJOB statement, and also that it is possible for more than one job to cite another job as a pre-requisite. In fact, SPANEX can handle an unlimited number of these relationships simultaneously, and, if you need to, you can define a very complex job network.

Generate the new RCM, using JCL similar to that shown in Figure 8 below.

```
//jobname  JOB  (accounting info),'SPANEX', . . .
//*  GENERATE SPANEX RCM FOR COMPLEX APPLICATION
//         EXEC RCMGEN,RCM=RCM200
//RCMASM.SYSIN  DD  *
JOB020   QUICKJOB
JOB040   QUICKJOB
JOB060   QUICKJOB
JOB100   QUICKJOB  PREREQ=(JOB020,JOB040)
JOB180   QUICKJOB  PREREQ=JOB100
JOB200   QUICKJOB  PREREQ=JOB100
JOB220   QUICKJOB  PREREQ=JOB060
JOB300   QUICKJOB  PREREQ=(JOB180,JOB200,JOB220)
JOB380   QUICKJOB  PREREQ=JOB300
JOB400   QUICKJOB  PREREQ=JOB300
JOB420   QUICKJOB  PREREQ=JOB300
RCM200   QUICKNET  LIBRARY=PDS
//RCMLKED.SYSLMOD  DD  DSN=SPANEX.TEST.RCMLIB(RCM200),
//       DISP=SHR
```

Figure 8. Generating the more complex RCM

Before this new job network can be run, you will need to create a JCL member for each of the new jobs. Using the example shown in Figure 5

on page 11, add JCL for jobs JOB020, JOB040, JOB060, JOB180, JOB220, JOB380, JOB400, and JOB420.

Now start the SPX TSO CLIST with the TSO command:

```
%SPX RCM200
```

and run this new network by issuing a SPANEX NETSTART command for network RCM200. Note that, as the job network runs, SPANEX will sometimes submit several jobs together, and will sometimes submit no jobs at all, depending on the job relationships you have defined in the RCM.

3.2 Tailoring the Job mix

Using SPANEX, the philosophy of defining job dependencies states that all jobs in an application system are defined together, whether or not each job executes every time that the application is run. For example, an application that is run every day is likely to have some jobs or functions that are required only weekly, or perhaps monthly, or even annually. Also, users often have ad hoc jobs, that are run only on request, based on factors that may be unknown to the operations department.

All these jobs are defined to SPANEX in the single RCM that represents this application. When the jobs are to be run, SPANEX is told what the particular job mix is for this execution. There is more than one way to do this, and SPANEX offers considerable flexibility in allowing the job mix information to be supplied.

For our first experience of this, we will use the job network defined in Figure 6 on page 16, and modify it before running by the use of some SPANEX command functions.

Ensure that you have run the network RCM200 successfully, through to completion. This will help to avoid the occurrence of problems with the next example. Use the SPANEX STATUS command for network RCM200, and check that the display shows the network as "COMPLETE".

Looking at the diagram on page 16, you will see that JOB060 has no pre-requisites, JOB100 is dependent upon both JOB020 and JOB040, and that JOB420 is the last job in the system and is dependent upon JOB300. All of these jobs are going to be omitted from the next run of the network. In this example, JOB060 is an ad hoc job, JOB100 is a weekly update that is run only on Fridays, and JOB420 produces monthly reports, and is run only on the last working day of each month.

To execute a Daily run of this application, these three jobs are omitted. We can do this with SPANEX without making any change at all to the definitions we have already completed. In fact, SPANEX can cope with

any combination of omitted jobs, and can even allow jobs to be omitted from an execution that is already part-way completed (provided, of course, that the job to be omitted has not yet started). SPANEX can also cope with a change of mind, and allows jobs to be re-included into the run.

Using the SPANEX full-screen TSO interface, as before, type the name of the first job to be omitted in the command input area and press the "EXCLUDE" function key: ie type

```
JOB060
```

and press PF5. This has the effect of generating a SPANEX EXCLUDE command for job JOB060, and this command effectively removes that job from the network for the next execution.

Repeat this process for jobs JOB100 and JOB420. Note that the effect of the EXCLUDE command can be undone by the use of the "INCLUDE" command or the "DELETE" command. You need not experiment with these other commands at this stage.

Now start the network with command:

```
NETSTART NOCONFRM
```

The NOCONFRM parameter is an optional keyword of the NETSTART command, that prevents SPANEX from issuing the SPX857A message prompt.

This page intentionally left blank.

4 Further SPANEX Facilities

4.1 The Global Log

An important aspect of implementing an automatic scheduling system such as SPANEX is being able to track back over actions and decisions the system has taken. This is essential to problem solving and to determining whether the definitions that have been made are correct.

SPANEX does not alter any existing reporting mechanisms that may be in place for the MVS system. For example, any SMF reports that are produced will not be changed or affected by a simple implementation of SPANEX.

SPANEX, however, adds some new facilities to aid in reporting and problem determination. For each execution of SPANEX, whether as a batch jobstep or within a TSO session, SPANEX produces a Message Log whenever it is provided with a SPXPRINT DD statement. This Message Log contains progress messages for the SPANEX execution, including confirmation of options selected, and a copy of each console information or error message produced by SPANEX.

Another source of SPANEX log information is the Global Log dataset. This is a sequential dataset that is specified on a Job Network basis, and which contains a chronological record of all scheduling actions, errors and commands processed for the Network. This log dataset may optionally be shared between Job Networks and may or may not be cleared of messages as each new execution of the Network begins.

We will now modify our test system to include a Global Log dataset. First, we must allocate disk space for the dataset. Run a job similar to the one shown in Figure 9 below to create the Global Log dataset. It is important that enough disk space is allocated to a Global Log dataset, as, if the space is filled, SPANEX will stop processing for the Job Network until the dataset is expanded. This is to ensure that logging of significant events can not be lost.

Note that, as before, this JCL may need to be modified so that the dataset name conforms to your installation standards. Remember the dataset name you choose, as this will later be specified in the RCM generation.

```
//jobname  JOB  (accounting info),'SPANEX', . . .
//*  ALLOCATE SPANEX GLOBAL LOG
//         EXEC PGM=IEFBR14
//GLOG     DD   DSN=SPANEX.TEST.GLOG,
//         UNIT=SYSDA,DISP=(,CATLG),
//         DCB=(RECFM=F,BLKSIZE=132),
//         SPACE=(CYL,(2,1))
//
```

Figure 9. Allocating a Global Log dataset

We must now modify the RCM to include the specification of the new Global Log dataset. Change your RCM generation input as shown in Figure 10 below, and re-run the generation process so that SPANEX will know the dataset name to use. SPANEX uses Dynamic Allocation for the Global Log, so no further changes need be made in order to implement it.

Lines that have not been changed since the previous version of the RCM are shown in faint type.

```
JOB020   QUICKJOB
JOB040   QUICKJOB
JOB060   QUICKJOB
JOB100   QUICKJOB  PREREQ=(JOB020,JOB040)
JOB180   QUICKJOB  PREREQ=JOB100
JOB200   QUICKJOB  PREREQ=JOB100
JOB220   QUICKJOB  PREREQ=JOB060
JOB300   QUICKJOB  PREREQ=(JOB180,JOB200,JOB220)
JOB380   QUICKJOB  PREREQ=JOB300
JOB400   QUICKJOB  PREREQ=JOB300
JOB420   QUICKJOB  PREREQ=JOB300
RCM200   QUICKNET  LIBRARY=PDS,                          X
             GLOGDSN=SPANEX.TEST.GLOG,GLOGDD=GLOG200
```

Figure 10. RCM definition statements including Global Log

Note that two parameters need to be specified to define a Global Log, the GLOGDSN and GLOGDD parameters. GLOGDSN specifies the full dataset name of the Global Log dataset; the GLOGDD parameter specifies a DDNAME which may be used by the SPANEX Utility, but which must specify a value unique to each Global Log dataset, as this is used as a serialization value by SPANEX.

Run the Job Network again, excluding jobs as specified in section 3.2 on page 18. During or after this run of the network, you should be able to browse the Global Log. SPANEX provides the TRACE command for this purpose. The TRACE command allows selection of records to view based on jobname, time range, or any other character string value, or combinations of these criteria. Look up this command in the SPANEX

manuals, or Reference Card, or in the SPANEX TSO online Help data, and experiment with it so that you become familiar with searching the Global Log for information.

4.2 Job Process Options

SPANEX provides some options that may be specified for individual jobs that alter the way the jobs are run by SPANEX or which request additional functions. These options are known as Job Process Options and any combination of the available options may be specified for any job.

You should refer to the SPANEX Restart and Job Networking Guide for a full description of all the options available. For the purposes of these worked examples, we will be using the Wait-For-Input option (PROCESS=WFI).

SPANEX Job Process Options are specified in the RCM, and we will be able to add this functionality to our Job Network without changing our jobs or JCL in any way.

Lines that have not been changed since the previous version of the RCM are shown in faint type.

```
JOB020   QUICKJOB
JOB040   QUICKJOB  PROCESS=WFI
JOB060   QUICKJOB
JOB100   QUICKJOB  PREREQ=(JOB020,JOB040)
JOB180   QUICKJOB  PREREQ=JOB100
JOB200   QUICKJOB  PREREQ=JOB100
JOB220   QUICKJOB  PREREQ=JOB060
JOB300   QUICKJOB  PREREQ=(JOB180,JOB200,JOB220)
JOB380   QUICKJOB  PREREQ=JOB300
JOB400   QUICKJOB  PREREQ=JOB300
JOB420   QUICKJOB  PREREQ=JOB300
RCM200   QUICKNET  LIBRARY=PDS,                          X
               GLOGDSN=SPANEX.TEST.GLOG,GLOGDD=GLOG200
```

Figure 11. RCM definition statements including Job Process Options

Re-generate the RCM as before. Then run the job network again, and note that, when JOB040 is ready to run, SPANEX issues message SPX893A to request information as to whether or not the required input for the job has arrived.

You can reply either "YES" or "NO" to this message. Select your reply, and type it into the command input area of the SPANEX screen, and then press "Enter". If you reply "YES", then the job will be submitted as usual.

If you reply "NO", the job will not be submitted. The rest of the network will continue to execute until the completion of JOB040 is a requisite. When you are ready to run job JOB040, you must issue a SPANEX "SCHEDULE" command for the job. Using the SPANEX formatted TSO screen, enter the jobname (JOB040) in the command input area, and press the "SCHEDULE" key (PF6). SPANEX will now submit the job, and execution of the network will continue.

Note that all of these events, messages and commands are documented in the Global Log dataset.

4.3 Ensuring Jobs complete successfully

One of the more important aspects of an automatic scheduling and restart system is a powerful means of determining reliably whether or not each job and jobstep has completed successfully. It is necessary to recognize Abends and also invalid Condition Codes.

Of course, this is not necessarily as simple as checking every step for Condition Code zero. There are many situations where user programs or utility programs need to issue non-zero completion codes. This is the whole basis for the Condition Code processing capabilities of MVS JCL. It is also common for failures in specific programs or jobsteps to be tolerable within the particular processing context of an application.

SPANEX allows all these cases to be catered for, and provides some very powerful tools for interrogating condition code values and checking them.

As an illustration, we will take a simple example of a jobstep that produces a non-zero condition code that is acceptable (ie that does not imply that the program has failed). To do this, we will take one of the jobs of our demonstration network and modify it so that it contains an incorrect execution of a standard MVS utility program. We will use an option of SPANEX known as Retrospective Condition Code Checking, whose function is to ensure that jobs complete successfully.

Take the JCL for JOB200 and replace it using the statements shown below.


```
//jobname  JOB  (accounting info),'SPANEX', . . .
//*  REPLACE APPLICATION JCL IN SPANEX JCL LIBRARY
//         EXEC PGM=IEBUPDTE,PARM=NEW
//SYSUT2   DD   DSN=SPANEX.TEST.RCMLIB,DISP=OLD
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DATA
./ ADD NAME=JOB200
//JOB200   JOB  (accounting info),'SPANEX', . . .
//STEP1    EXEC PGM=IEFBR14
//STEP2    EXEC PGM=IEBGENER
//
/*
```

Figure 12. JCL to produce a non-zero Condition Code

Modify the RCM as shown in Figure 13 below. This adds the Retrospective Condition Code Checking function.

Lines that have not been changed since the previous version of the RCM are shown in faint type.

```
JOB020   QUICKJOB
JOB040   QUICKJOB  PROCESS=WFI
JOB060   QUICKJOB
JOB100   QUICKJOB  PREREQ=(JOB020,JOB040)
JOB180   QUICKJOB  PREREQ=JOB100
JOB200   QUICKJOB  PREREQ=JOB100,SCANOPT=3
JOB220   QUICKJOB  PREREQ=JOB060
JOB300   QUICKJOB  PREREQ=(JOB180,JOB200,JOB220)
JOB380   QUICKJOB  PREREQ=JOB300
JOB400   QUICKJOB  PREREQ=JOB300
JOB420   QUICKJOB  PREREQ=JOB300
RCM200   QUICKNET  LIBRARY=PDS,                          X
               GLOGDSN=SPANEX.TEST.GLOG,GLOGDD=GLOG200
```

Figure 13. RCM definition statements including Condition Code Checking

Now run the network again, and note that JOB200 fails because of the non-zero condition code in STEP2. The network execution will stop at this point, and SPANEX will require that the execution of JOB200 is corrected before any further processing will be allowed.

However, in this case, STEP2 of JOB200 can issue a high condition code without this representing an error, so we need to adjust our RCM definition to inform SPANEX of this.

Change the RCM again as shown in Figure 14 below, and re-generate it.

Once again, lines that have not been changed since the previous version of the RCM are shown in faint type.

```
JOB020   QUICKJOB
JOB040   QUICKJOB  PROCESS=WFI
JOB060   QUICKJOB
JOB100   QUICKJOB  PREREQ=(JOB020,JOB040)
JOB180   QUICKJOB  PREREQ=JOB100
JOB200   QUICKJOB  PREREQ=JOB100,SCANOPT=3
STEP2    QUICKSTP  ACCRC=16
JOB220   QUICKJOB  PREREQ=JOB060
JOB300   QUICKJOB  PREREQ=(JOB180,JOB200,JOB220)
JOB380   QUICKJOB  PREREQ=JOB300
JOB400   QUICKJOB  PREREQ=JOB300
JOB420   QUICKJOB  PREREQ=JOB300
RCM200   QUICKNET  LIBRARY=PDS,                          X
               GLOGDSN=SPANEX.TEST.GLOG,GLOGDD=GLOG200
```

Figure 14. RCM definition statements including Allowable Return Code

The change made here is to define explicitly the step of JOB200 that is causing the problem. The QUICKSTP statement accomplishes this. The ACCRC= parameter of the QUICKSTP statement specifies a Return Code threshold for the program that executes in this jobstep. Any condition code issued by this program, from zero up to and including the value specified here, will be treated by SPANEX as a good completion of the program. In our case, with an execution of IEBGENER with no DD statements, the condition code issued is within the allowable range we have specified, and so the next execution of this jobstep should be treated by SPANEX as successful.

SPANEX also provides a more sophisticated method for validating condition codes, whereby any specific value or combination or values can be checked for, rather than just using a threshold technique. This is implemented by a supplied user exit routine called SPXUCHEK. This is fully documented in the SPANEX General Usage manual.

5 Adding Calendar Processing to the Application

This chapter begins by describing the use of manual SPANEX procedures for the performing of calendar-related scheduling functions. The techniques described and practised are of use in all SPANEX installations, but some of them can be automated by the use of the SPANEX Automatic Calendar facility. If you would like to skip over the sections on manual SPANEX calendars, please continue with Section 5.3 on page 29.

5.1 Defining Daily Schedules for Manual Calendars

When, in section 3.2, we issued a series of SPANEX "EXCLUDE" commands to modify the mix of jobs in our application, we were performing a task that probably would need to be performed for each run of the network. In most cases, not all jobs of an application are run every time the application is run.

However, usually there is a limited number of different job combinations that is run for any given application. There may be a Daily run, a Weekly run, a Month-end run, and so on. The selection of the correct run for a particular application execution is performed by a function known as a Calendar.

For our example application, we will set up series of schedules or calendars to represent some simple combinations of jobs. As we saw before, JOB060 is an ad hoc job, JOB100 is run only on Fridays, and JOB420 is run only at accounting month-end. All the other jobs in the network execute for every network run.

Run a job similar to that shown in Figure 15 below. This will add manual calendars to the SPANEX Utility Command Library, with member names that are descriptive of the days on which they are to be run.

```
//jobname  JOB  (accounting info),'SPANEX', . . .
//*  ADD CALENDARS TO SPANEX UTILITY LIBRARY
//         EXEC PGM=IEBUPDTE,PARM=NEW
//SYSUT2   DD   DSN=SPANEX.TEST.UTILLIB,DISP=OLD
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
./ ADD NAME=DAILY
  EXCLUDE JOB=JOB060
  EXCLUDE JOB=JOB100
  EXCLUDE JOB=JOB420
  NETSTART NOCONFRM
./ ADD NAME=FRIDAY
  EXCLUDE JOB=JOB060
  EXCLUDE JOB=JOB420
  NETSTART NOCONFRM
./ ADD NAME=MONTHEND
  EXCLUDE JOB=JOB060
  NETSTART NOCONFRM
/*
```

Figure 15. Adding Manual SPANEX Calendars to the Utility Library

To use one of these defined calendars requires an absolute minimum of keystrokes. As an example, we will now execute once again the "Daily" run of our application.

Start the TSO SPANEX facility with the command:

```
%SPX RCM200
```

as before. To execute the Daily run, type the name:

```
DAILY
```

in the command input area and press the "INPUT" function key (PF8).

SPANEX will read in the command sequence from the "DAILY" member of the Utility Command Library, and execute the commands in sequence. This will automatically remove from the network all the jobs that do not run on a daily basis, and start the execution of the application.

Try the other calendars we have defined (FRIDAY and MONTHEND) to show the other job combinations.

5.2 Configuring a Calendar-based Application

The examples shown in the previous section illustrate one of the ways that calendar-based applications can be controlled by SPANEX, using manual methods.

SPANEX has many ways of helping with this problem, and these can be evaluated for each job or each application that is being implemented. There are no restrictions on using different techniques or different SPANEX commands in any combination.

For example, jobs that are not generally run as part of a network can be defined with the "PROCESS=EXCLUDE" option, which means that the job is excluded from the Network by default. When the job is required, it can be reinstated by means of the SPANEX "INCLUDE" command. Another approach might be to allow user department personnel to issue EXCLUDE and INCLUDE commands (perhaps disguised within user-friendly TSO CLISTs) to configure their own applications, or to request ad hoc jobs without needing to involve the operations team.

5.3 Defining Automatic SPANEX Calendar Processing

The SPANEX system has comprehensive facilities for automating the calendar processing shown above. Although this requires a little more effort to set up initially, it can allow each application to be automatically configured for the day it is being run, without any manual intervention at all.

We will now modify our previous example job network to include automatic SPANEX calendars. This section does not cover the definition of the SPANEX System Calendar tables; this function is performed for the current calendar year when SPANEX is installed, and it is a normal SPANEX maintenance function to ensure that the Calendar tables are kept up-to-date.

Using our sample application, we will automate the calendar processing that we performed manually in Section 5.1 above. We will define the application to run only on weekdays (Monday to Friday), and we will set JOB060 as an ad hoc job, JOB100 as a job run only on Fridays, and JOB420 as a job run only at month-end. For the purposes of this exercise, we will define month-end as the fourth Friday in the month; this allows us to use a standard SPANEX Calendar "day-type" and avoids the necessity of determining an actual month-end date.

Change the RCM again as shown in Figure 16 below, and re-generate it.

Once again, lines that have not been changed since the previous version of the RCM are shown in faint type.

Note that we have set the ad hoc job, JOB060, as "PROCESS=EXCLUDE". This means that the job will always be excluded from the application, unless we take some action to include it. Thus this is a true "ad hoc" job. Should the job be required to be run, a SPANEX INCLUDE command should be used before the execution of the application passes JOB060. If the INCLUDE command is to be used before the Job Network is started, it should include the "NEXT" option.

```
JOB020   QUICKJOB
JOB040   QUICKJOB  PROCESS=WFI
JOB060   QUICKJOB  PROCESS=EXCLUDE
JOB100   QUICKJOB  PREREQ=(JOB020,JOB040),                X
               RUNDAYS=FRIDAY
JOB180   QUICKJOB  PREREQ=JOB100
JOB200   QUICKJOB  PREREQ=JOB100,SCANOPT=3
STEP2    QUICKSTP  ACCRC=16
JOB220   QUICKJOB  PREREQ=JOB060
JOB300   QUICKJOB  PREREQ=(JOB180,JOB200,JOB220)
JOB380   QUICKJOB  PREREQ=JOB300
JOB400   QUICKJOB  PREREQ=JOB300
JOB420   QUICKJOB  PREREQ=JOB300,RUNDAYS=4THFRI
RCM200   QUICKNET  LIBRARY=PDS,                           X
               NONDAYS=(SATURDAY,SUNDAY),                 X
               GLOGDSN=SPANEX.TEST.GLOG,GLOGDD=GLOG200
```

Figure 16. RCM definition statements including Automatic Calendars

Having defined our application with calendar specifications, we merely have to ensure that a NETSTART command is issued for the Job Network every day. If the day happens to be a Saturday or Sunday, the NETSTART will have no effect, because Saturday and Sunday are defined as "NONDAYS" on the QUICKNET statement. On other days, the application will be automatically configured for the correct combination of jobs.

Issue a NETSTART command for the Job Network RCM200 again, and note the effect of the Automatic Calendar processing.

6 JCL Considerations for New SPANEX Users

6.1 The Effect of SPANEX on Existing JCL

SPANEX need have no effect at all on existing working customer JCL. All existing JCL for batch jobs will execute unaltered when using SPANEX Job Scheduling. The only limitation is that JCL must be stored on a library dataset (Partitioned DataSet (PDS or PDSE), CA-LIBRARIAN or CA-PANVALET), with one library member per job. This is frequently already the case, and, if not, it is usually a simple exercise to set up if JCL is currently stored in a different format.

6.2 The Effect of SPANEX on New JCL

SPANEX does not necessarily change the way JCL is written in a customer installation. If the customer wishes to continue his existing JCL standards and techniques, then SPANEX will not impose any restrictions on this, given the limitation discussed in Section 6.1 above. However, the implementation of SPANEX lifts many practical restrictions that exist as a result of the lack of automatic job checking and control that SPANEX will provide.

For example, now that jobs are automatically submitted, and condition code verification is performed automatically, there is no effective limit on the number of jobs that can be run for an application system. There is no longer a need to keep to a limited number of jobs in order to make the system operable. This is turn means that large complex jobs, executing many steps and with many step condition code dependencies, can become a thing of the past. JCL can be made much simpler, and much more use can be made of MVS multi-programming, which will reduce the elapsed time taken for an application to run. If there are no direct dependencies between programs in a application, then why not run them at the same time? The sample job given below provides an illustration of this. Note that we do not necessarily recommend that customers immediately start rewriting all their application systems as soon as SPANEX is installed. The change to this new philosophy should be made gradually, with application jobs being restructured whenever they are being changed for other reasons. Of course, the benefits realised by the customer as soon as the SPANEX technique has been tried, may persuade him to change his existing applications sooner rather than later.

```
Step 1            Allocate disk space

Step 2            Validate user input

Step 3            Process output file from another
                  application, produce file for Step 5

Step 4            Read database, produce file for Step 6

Step 5            Update database with output from Step
                  3, produce file for Step 6

Step 6            Update database, using files from Step
                  3 and Step 4

Step 7            Produce reports for user department

Step 8            Produce reports for accounts
                  department

Step 9            Produce output file from database for
                  passing to next application

Step 10           Delete work disk files
```

Figure 17. A Hypothetical Job Before SPANEX

When SPANEX automatic scheduling is available, this sequence of jobsteps could be reorganised into multiple jobs as shown below:

```
Job 1       Allocate disk space

Job 2       Validate user input

Job 3       Process output file from another
            application, produce file for Job 5

Job 4       Read database, produce file for Job 6

Job 5       Update database with output from Job 3, pro-
            duce file for Job 6

Job 6       Update database, using files from Job 3 and
            Job 4

Job 7       Produce reports for user department

Job 8       Produce reports for accounts department

Job 9       Produce output file from database for
            passing to next application.  Delete work
            disk files
```

Figure 18. Splitting into Multiple Jobs

Now, using SPANEX automatic scheduling and job dependency control, the sequence of jobs can be defined to SPANEX as a Network, and the diagram of relationships is shown in Figure 18 below:

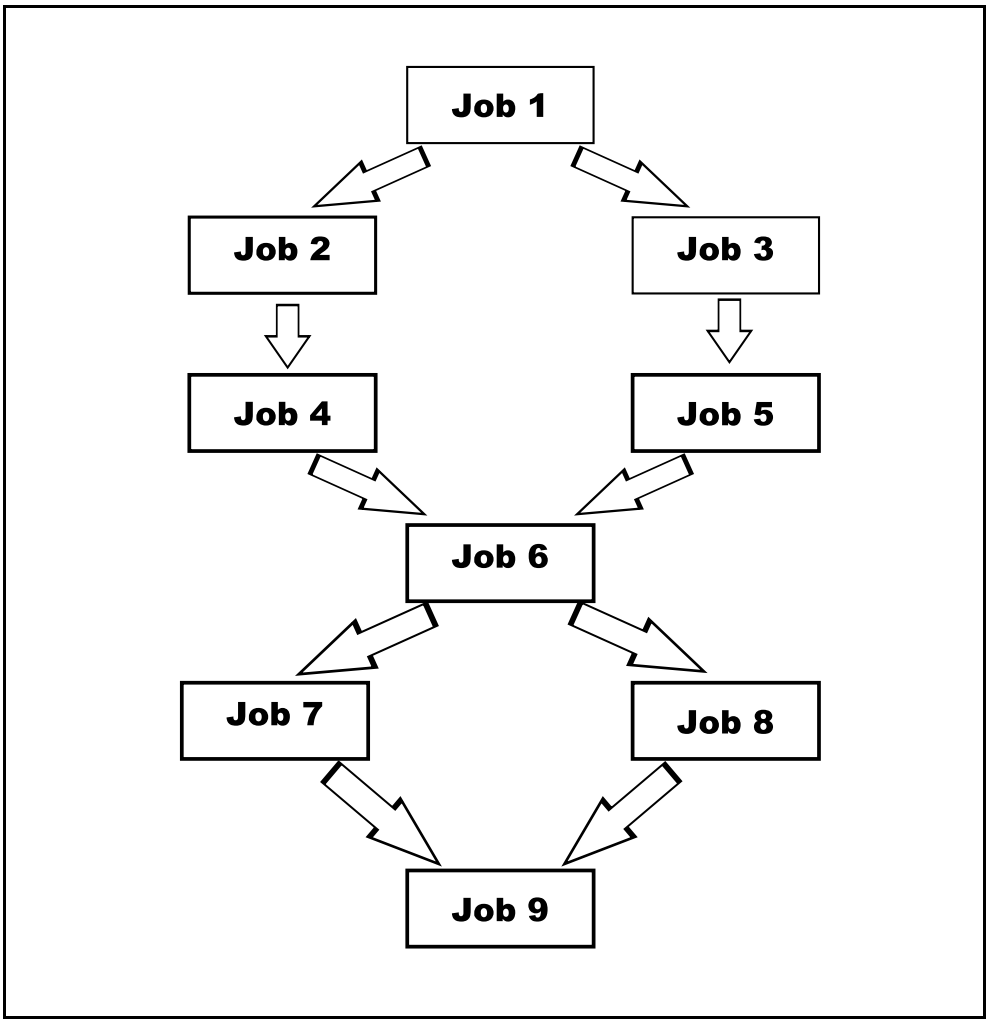

Diagram 4. Network Diagram of Derived Jobs

The diagram (Figure 18) shows the flow of control through the jobs of the application. Where jobs are shown in parallel, they can actually execute in parallel (at the same time, assuming there are available MVS Initiators). Therefore, in a typical system, the total elapsed running time of the application can be significantly reduced. There are, naturally, no SPANEX-defined limits on the number of jobs that may be created, nor on the complexity or number of threads of execution of an application. Note that even this approach does not require any significant JCL change - only the splitting of jobs at existing step boundaries.

6.3 Special SPANEX Features available if JCL is changed

Although no JCL changes are required in order to use SPANEX for automatic job scheduling, many additional SPANEX features are implemented by embedding executions of SPANEX within batch jobs. The most powerful and far-reaching of these features is SPANEX Automatic Job Restart, whereby extra jobsteps are built-in to jobs to provide automated recovery from predictable types of job failure. For example, a recovery step might be provided to reload a dataset that is updated during the job - SPANEX can detect when the step that performs that update has failed at a previous attempt, and will automatically select and execute the recovery step the next time the job is submitted.

An example is given here of the JCL required to implement a self-recovering job using SPANEX Automatic Job Restart. This example is adapted from an example in the SPANEX Restart and Job Networking Guide.

This job contains three application job steps, named STEP1, STEP2 and STEP3. STEP1 runs a user program to perform input validation; STEP2 performs a master file (or database) update based on the output from STEP1; and STEP3 runs a report program. Should STEP2 fail during execution, a utility must be run to restore the master file that has been partially updated by the failing job.

The SPANEX philosophy is always to perform recovery actions as part of the following attempt to run the job. Therefore, when a job fails, no attempt is made to recover at that time. This would occur in any case if the reason for the job crash was a power failure, MVS failure, or operational error. The SPANEX recovery process is executed the next time the job is run. Partial JCL for this example is shown below. Program names, DD statements, etc, are omitted, and only the SPANEX requirements are shown. Note that the user application programs are run under the control of SPANEX, and that SPANEX is the program named in the PGM= parameter of the EXEC statement in each case, even for an application jobstep: the name of the user program is passed to SPANEX via the PARM field.

```
//jobname JOB  (account ...),CLASS=x, ...
//SPXINIT EXEC PGM=SPANEX,PARM=',OPT=I,NET=netname'

//RSTEP2  EXEC PGM=SPANEX,PARM='utility,OPT=M/utility-parm',
//             COND=(8,NE,SPXINIT)
                    .
            user DD statements for utility
                    .
//STEP1   EXEC PGM=SPANEX,PARM='userpg1,OPT=M/userpg1-parm',
//             COND=(0,NE,SPXINIT)
                    .
            user DD statements for userpg1
                    .
//STEP2   EXEC PGM=SPANEX,PARM='userpg2,OPT=M/userpg2-parm',
//             COND=(8,LT,SPXINIT)
                    .
            user DD statements for userpg2
                    .
//STEP3   EXEC PGM=SPANEX,PARM='userpg3,OPT=M/userpg3-parm'
                    .
            user DD statements for userpg3
                    .
```

Figure 19. SPANEX Automatic Job Restart

In addition to the JCL changes, and in common with all uses of SPANEX for Job Scheduling, SPANEX Restart requires the generation, by the user, of a Restart Control Module or RCM. For the SPANEX Restart system, this RCM defines the sequence of jobsteps to be used by SPANEX for this job. This is used by the SPXINIT step in the example job, which examines the results of the previous execution of the job, and then terminates with a Condition Code that selects the remaining steps of the job to be run.

In this example, it works like this. For a "clean start", or normal run, of the job, the SPXUTIL step issues cond code 0, and steps STEP1, STEP2 and STEP3 execute in sequence. For a restart after a failure of STEP1, the same sequence occurs, SPXUTIL issues a cond code 0, and steps STEP1, STEP2 and STEP3 execute (STEP1 performs only an input validation process, and this step can be rerun as necessary).

However, should a failure have occurred in STEP2, some recovery action is necessary, and SPXUTIL terminates with cond code 8. This causes step RSTEP2 to be run (the recovery utility for the file updated by STEP2), and then other steps run are STEP2 and STEP3. Should a failure have occurred in STEP3, then SPXUTIL will issue cond code 12, and only STEP3 will be run during the restart.

It is also possible for the recovery step, RSTEP2, to fail. Even this is covered by this example. For a restart after a failure in RSTEP2, the SPXUTIL step will terminate with cond code 8, causing another attempt to be made at running the recovery utility, followed by steps STEP2 and STEP3.

SPANEX, as usual, places no limitations on the complexity of use of this restart facility. Many user programs, and many recovery steps can be built into each job. We always recommend the simplest possible use, however, and this usually involves splitting large jobs into many smaller jobs, and then using SPANEX Job Scheduling to control the sequence of jobs and their submission for processing.

6.4 Further SPANEX Features implemented via JCL parameters

There are many other SPANEX features that are selected by JCL parameters options and by executing SPANEX within user jobs. These features are not related to the Job Scheduling or Job Restart features of SPANEX, but can of course be used in combination with them. These features are all documented in the SPANEX General Usage Manual, and are too many to be listed here. However, such features include many powerful facilities for checking for the successful execution of a user program, and for determining the action to be taken should a user program fail (this can include requesting an acknowledgement from the operator that he has noticed the failure).

This page intentionally left blank.

7 Cross-Application Relationships

The SPANEX Utility is invoked when "OPT=U" is specified as a SPANEX parameter in a batch or TSO execution, or via one of the Extended TP support modules. The SPANEX Utility provides the ability for the user or the system operator to display or modify the status of any SPANEX-controlled job, and also provides the main vehicle for exercising control over SPANEX job networks. The modification of restart status can be prevented for jobs within a given RCM by specifying "OPTU=NO" on the SPXRCM or QUICKNET macro, although this does not prevent the displaying of status information.

An invocation of the SPANEX Utility, to perform any desired SPANEX control function, can be embedded as a jobstep within an application batch job at any time. This provides a mechanism whereby other SPANEX jobs and applications can be influenced.

For a simple example, consider an application job (Job A) that creates a file that is used as input by another application system (Job B in System B). Obviously, Job B cannot be run until Job A has successfully created the file that Job B is to read. However, there is no reason that other jobs in System B, that occur before Job B, should not be run. If we can implement an automatic method of notifying System B that the file is ready we can eliminate manual intervention altogether.

In order to achieve this with SPANEX, we might take the following steps. First, Job B should be defined with a SPANEX Event. Part of a QUICKJOB statement that achieves this is shown below:

```
JOBB     QUICKJOB  HOLD=1,PREREQ=( ...
```

Note the "HOLD=1" parameter - this indicates that SPANEX Event No 1 must be posted complete before JOB B can be run. This does not affect the normal SPANEX job pre-requisite processing.

In addition, we need to add to JOB A, after the jobstep that creates the file, an invocation of the SPANEX Utility to post the completion of Event 1 for JOB B. An example of this is shown below:

```
//SPXUTIL  EXEC  PGM=SPANEX,PARM=',OPT=U'
//SPXPRINT DD    SYSOUT=A
//TASKLIB  DD    DSN=SPANEX.TEST.RCMLIB,DISP=SHR
//JOBPDS   DD    DSN=SPANEX.TEST.JCLLIB,DISP=SHR
//SPXRSCTL DD  *
  POST  NET=SYSTEMB,JOB=JOBB,EVENT=1
/*
```

When the POST command is executed, Event 1 for JOB B will be set complete, and, if all JOB B's pre-requisites have completed, it will be submitted by SPANEX for execution.

Use of SPANEX Events for inter-system dependencies can be made much more elaborate than this, if you need to. Each job can have up to eight events, and events can be set incomplete dynamically as well (using the HOLD command). If a job has to wait for more than one input file, for example, then each file could be represented by a different event. Only when all events for a job are complete will a job be submitted by SPANEX for execution.

Now that you have gained familiarity with some of the features of SPANEX, as an exercise create two applications Job Networks, and inter-link them using SPANEX Events. Test your definitions by running both Networks simultaneously, allowing some of the Events to be automatically POSTed.

8 Conclusions

An examination of the other manuals for SPANEX will show that this introductory document has merely scratched the surface of the facilities offered by SPANEX.

SPANEX is an extremely powerful and flexible system for scheduling and control of jobs, and yet is simple and quick to use and to understand.

If you have worked through the examples in this manual, you will gained an insight into how SPANEX performs its basic functions. We recommend that after achieving familiarity with the features introduced by these examples, you read the SPANEX Restart and Job Networking Guide to learn about the many other facilities and the other ways of achieving automatic scheduling using SPANEX.

Then you can determine the design and implementation of automation that best suits your environment. Note always that different jobs or applications can use differing SPANEX features and techniques if this is appropriate.

This manual has not attempted to document the SPANEX Job Restart facility, the use of CA-PANVALET or CA-LIBRARIAN, or many dozens of SPANEX features for the execution of batch or TSO jobsteps and programs.

We wish you every success with the automation of your jobs and application systems using SPANEX.

This page intentionally left blank.

This manual is published by

Span Software Consultants Limited

Little Moss, Peacock Lane

High Legh

Knutsford

Cheshire

WA16 6PL

England

Tel: +44/0 1565 832999

Fax: +44/0 1565 830653

Email: spanex@spansoftware.com

www.spansoftware.com

to whom all comments and suggestions should be sent.