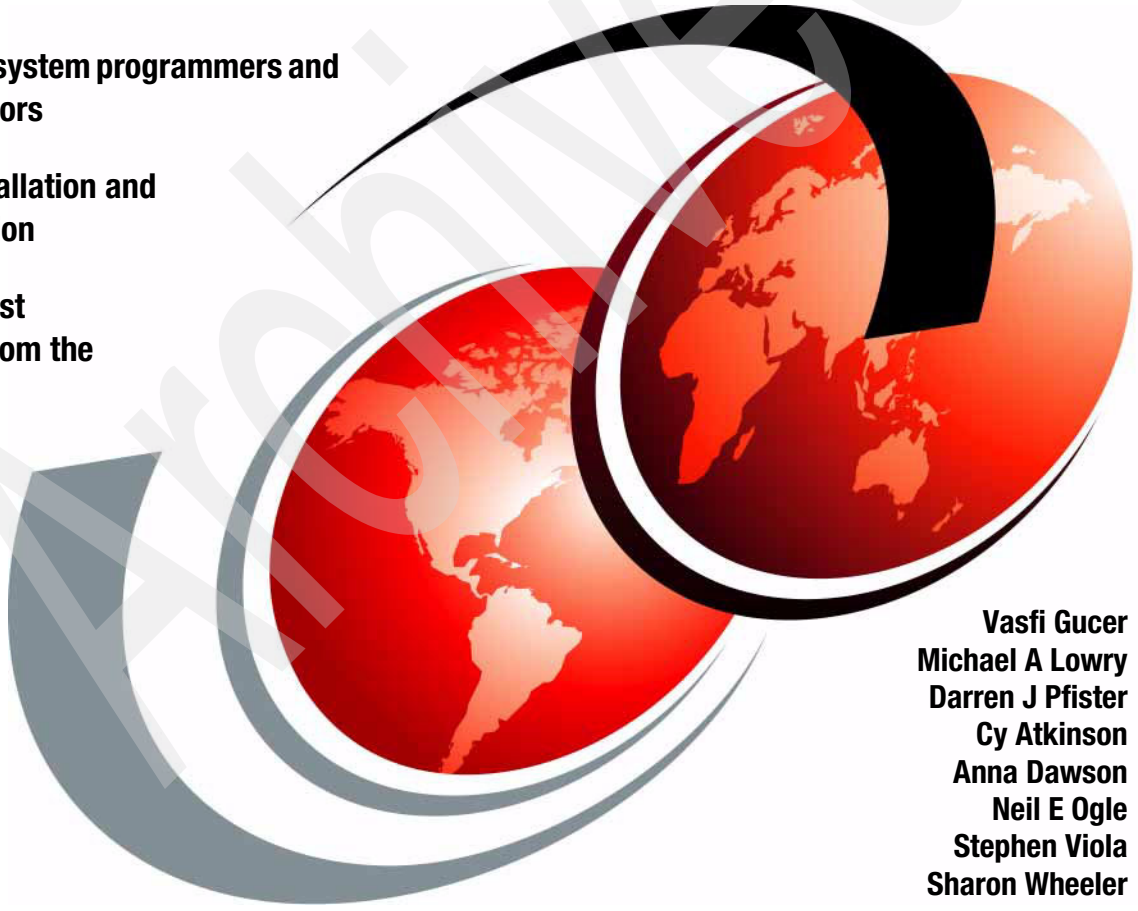
Tivoli software

IBM

IBM Tivoli Workload Scheduler for z/OS Best Practices

End-to-end and mainframe scheduling

- A guide for system programmers and administrators
- Covers installation and customization
- Includes best practices from the



Vasfi Gucer
Michael A Lowry
Darren J Pfister
Cy Atkinson
Anna Dawson
Neil E Ogle
Stephen Viola
Sharon Wheeler

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM Tivoli Workload Scheduler for z/OS Best Practices - End-to-end and mainframe scheduling

May 2006

SG24-7156-01

Note: Before using this information and the product it supports, read the information in “Notices” on page xv.

Second Edition (May 2006)

This edition applies to IBM Tivoli Workload Scheduler for z/OS Version 8.2.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
AS/400®
CICS®
DB2®
Hiperbatch™
HACMP™
IBM®
IMS™
Language Environment®
Maestro™
MVS™
NetView®
OS/2®
OS/390®
OS/400®
pSeries®
RACF®
Redbooks™
Redbooks (logo) ™
S/390®
Sequent®
Systems Application Architecture®
SAA®
Tivoli Enterprise™
Tivoli Enterprise Console®
Tivoli Management Environment®
Tivoli®
TME®
VTAM®
WebSphere®
z/OS®
zSeries®

The following terms are trademarks of other companies:

Java, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, PowerPoint, Windows server, Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbook is a reference for System Programmers and Administrators who will be installing IBM Tivoli® Workload Scheduler for z/OS® in mainframe and end-to-end scheduling environments.

Installing IBM Tivoli Workload Scheduler for z/OS requires an understanding of the started tasks, the communication protocols and how they apply to the installation, how the exits work, how to set up various IBM Tivoli Workload Scheduler for z/OS parameters and their functions, how to customize the audit function and the security, and many other similar topics.

In this book, we have attempted to cover all of these topics with practical examples to help IBM Tivoli Workload Scheduler for z/OS installation run more smoothly. We explain the concepts, then give practical examples and a working set of common parameters that we have tested in our environment.

We also discuss both mainframe and end-to-end scheduling, which can be used by IBM Tivoli Workload Scheduler for z/OS specialists working in these areas.

The team that wrote this redbook

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Vasfi Gucer is an IBM Certified Consultant IT Specialist working at the ITSO Austin Center. He worked with IBM Turkey for 10 years and has been with the ITSO since January 1999. He has more than 12 years of experience in systems management, networking hardware, and distributed platform software. He has worked on various Tivoli customer projects as a Systems Architect in Turkey and the United States. Vasfi is also a Certified Tivoli Consultant.

Michael A Lowry is an IBM-certified consultant and instructor based in Stockholm, Sweden. He has 12 years of experience in the IT services business and has been with IBM since 1996. Michael studied engineering and biology at the University of Texas. He moved to Sweden in 2000 and now holds dual citizenship in the United States and Sweden. He has seven years of experience with Tivoli Workload Scheduler and has extensive experience with IBM network and storage management products. He is also an IBM Certified AIX® Support Professional.

Darren Pfister is a Senior IT Specialist working out of the Phoenix, Arizona, office. He has worked for IBM for six years and is part of the z/Blue Software Migration Project. He has more than 12 years of experience in scheduling migrations, project management, and technical leadership. He has worked on various IBM Global Services customer accounts since joining IBM in 1999. He also holds a Masters degree in Computer Information Systems and is currently working on his PhD in Applied Management and Decision Sciences.

Cy Atkinson has been with IBM since 1977, providing hardware support to large systems customers in the Green Bay, Wisconsin, area until 1985 when he moved to San Jose and joined the JES2/OPC L2 support team. In 1990 he became OPC L2 team leader for the US, moving OPC support to Raleigh in 1993. Cy is a regular speaker in ASAP (Tivoli Workload Scheduler User's Conference).

Anna Dawson is a U.K.-based Systems Management Technical Consultant working at IBM Sheffield. Before joining IBM, she worked at a very large customer site, where she was the primary person responsible for the day-to-day customization, implementation, and exploitation of their batch scheduling environment. She has many years of experience with the Tivoli Workload Scheduler for z/OS product and has focused most recently on the area of performance.

Neil E Ogle is an Advisory IT Specialist - Accredited who works doing migrations from OEM products to the Tivoli Workload Scheduler product. He has 39 years of experience in IT system programmingm and his expertise includes TWS, z/OS, ADTOOLS, and JES2. Neil is a resident of Eureka Springs, Arkansas, and works remotely worldwide supporting customers.

Stephen Viola is an Advisory Software Engineer for IBM Tivoli Customer Support, based in Research Triangle Park, North Carolina. He is a member of the Americas Tivoli Workload Scheduler Level 2 Support Team. In 1997, he began to support Tivoli System Management software. Since 2003, he has worked primarily on Tivoli Workload Scheduler for z/OS, especially data store and E2E. His areas of expertise include installation and tuning, problem determination and on-site customer support.

Sharon Wheeler is a Tivoli Customer Support Engineer based in Research Triangle Park, North Carolina. She is a member of the Americas Tivoli Workload Scheduler L2 Support Team. She began working for IBM as a member of the Tivoli services team in 1997, joined the Tivoli Customer Support organization in 1999, and has supported a number of products, most recently TBSM. In 2004, she began working on the Tivoli Workload Scheduler for z/OS L2 Support team

Thanks to the following people for their contributions to this project:

Budi Darmawan
Arzu Gucer
Betsy Thaggard
International Technical Support Organization, Austin Center

Robert Haimowitz
International Technical Support Organization, Raleigh Center

Martha Crisson Art Eisenhour
Warren Gill
Rick Marchant
Dick Miles
Doug Specht
IBM USA

Finn Bastrup Knudsen
IBM Denmark

Antonio Gallotti
Flora Tramontano
IBM Italy

Robert Winters
Blue Cross of Northeastern Pennsylvania

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners, and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-7156-01
for IBM Tivoli Workload Scheduler for z/OS Best Practices - End-to-end and mainframe scheduling
as created or updated on May 16, 2006.

May 2006, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- Chapter 12 ”Using Tivoli Workload Scheduler for z/OS effectively” has been added.
- Part 2 “Tivoli Workload Scheduler for z/OS end-to-end scheduling” has been added.

Part 1

Tivoli Workload Scheduler for z/OS mainframe scheduling

In this part we introduce the installation of IBM Tivoli Workload Scheduler for z/OS and cover the topics either applicable only for mainframe scheduling or both end-to-end and mainframe scheduling. Topics that exclusively applicable to end-to-end scheduling will be covered in Part 2, “Tivoli Workload Scheduler for z/OS end-to-end scheduling” on page 307.

1

Tivoli Workload Scheduler for z/OS installation

When getting ready to install IBM Tivoli Workload Scheduler for z/OS, a System Programmer or Administrator must have an understanding of the started tasks, the communication protocols, and how they apply to the installation. This chapter is a guideline for the installation, and it points to other chapters in the book that explain how the different pieces of IBM Tivoli Workload Scheduler for z/OS work together, how the exits work, a starting set of parameters and their functions, the audit function, and many other items of interest.

As you can see, this is not just for “How do I install the product?” but is more geared toward the experienced System Programmer or Administrator who will need and use the chapters in this book to understand, install, verify, and diagnose problems, and use many of the features of the product. This chapter covers a basic installation of the Controller/Tracker/DataStore.

This chapter includes the following topics:

- Before beginning the installation
- Starting the install
- Updating SYS1.PARMLIB
- SMF and JES exits installation

- Running EQQJOBS
- Security
- Allocating the data sets
- Creating the started tasks
- Defining Tivoli Workload Scheduler for z/OS parameters
- Setting up the ISPF environment
- Configuring Tivoli Workload Scheduler for z/OS; building a current plan
- Building a workstation

1.1 Before beginning the installation

Before you begin the installation, take some time to look over this book, and read and understand the different chapters. Chapter 3, “The started tasks” on page 69 offers an explanation of how the product works and how it might be configured. You might want to read Chapter 6, “Tivoli Workload Scheduler for z/OS exits” on page 153 for an idea of what is involved as far as system and user exits. Although this installation chapter points you to certain areas in the book, it would be helpful to the person installing to read the other chapters in this book that apply to the install before beginning.

1.2 Starting the install

The installation of most IBM products for z/OS begins with the SMP/E (system modification program/extended) installation of the libraries. We do not cover the SMP/E install itself as it is widely covered in the *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264. Instead, we include the libraries from the output of the SMP/E job and their functions. These libraries normally have a prefix of Sysx.TWS82.SEQQxx.

The libraries are named AEQQxxx (DLIBs) and SEQQxxx (TLIBs) as seen in Table 1-1.

Table 1-1 Library names

DLIB	TLIB	Description
AEQQPNL0	SEQQPNL0	ISPF Panel library
AEQQMOD0	SEQQLMD0	Load library
AEQQMSG0	SEQQMSG0	Message library
AEQQMACR0	SEQQMAC0	Assembler macros
AEQQCLIB	SEQQCLIB	CLIST library
AEQQSAMP	SEQQSAMP	Sample exits, source code, and jobs
AEQQSKL0	SEQQSKL0	Skeleton library and Audit CLIST
AEQQTBL0	SEQQTBL0	ISPF tables
EQQDATA	SEQQDATA	Sample databases
AEQQMISC	SEQQMISC	OCL compiled library, DBRM files for DB2®

SEQQLMD0 load library must be copied into the linklist and authorized.

When EQQJOBS has been completed, one of the libraries produced is the Skeleton Library. You should modify the temporary data sets of the current and long-term plan member skeletons (EQQDP*,EQQL*), increasing their size (100 Cyl. is a starting point) depending on your database size. The Audit CLIST in the Skeleton library (HLQ.SKELETON(EQQAUDNS), which is generated by EQQJOBS Option 2), must be modified for your environment and copied to your CLIST library.

Note: The Tivoli Workload Scheduler for z/OS OCL (Control Language) is shipped as COMPILED REXX and requires the REXX/370 V1R3 (or higher) Compiler Library (program number 5696-014).

Chapter 3, “The started tasks” on page 69, refers to the started tasks, their configuration, and purpose in life. It will be beneficial to read this and understand it prior to the install. You can find additional information about started task configuration in *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264. This is also covered in detail in Chapter 4, “Tivoli Workload Scheduler for z/OS communication” on page 87. This chapter should also be read before installing because it helps you decide whether you want to use XCF or VTAM® as an access method.

DataStore is an optional started task, but most Tivoli Workload Scheduler for z/OS users install it because it is necessary for restarts and browsing the sysout from Tivoli Workload Scheduler. Therefore, it is covered in this install procedure and not as a separate chapter. It also is covered in the *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265.

The Sys1.Parmlib changes and SMF/JES (system measurement facility/job entry subsystem) exit changes require an IPL so it seems appropriate to do those steps as soon as possible, because most systems are not IPLed frequently, and other steps can be done while waiting for an IPL.

Note: You can use the following link for online access to IBM Tivoli Workload Scheduler for z/OS documentation:

`http://publib.boulder.ibm.com/tividd/td/WorkloadScheduler8.2.html`

1.3 Updating SYS1.PARMLIB

The parmlib definitions can be classified into seven tasks:

- Update the IEFSSNxx member
- Updating the IEAAPFxx member
- Updating the SMFPRMxx member
- Update Dump definitions
- Update the XCF options
- Update IKJTSOxx member
- Update SCHEDxx member

There are other, optional parmlib entries, which are described in *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264.

1.3.1 Update the IEFSSNxx member

The IEFSSNxx member is the member that controls subsystems in z/OS. Tivoli Workload Scheduler for z/OS is using three primary subsystems so it requires two entries in this member (one for the Tracker and one for the Controller). The parameter that can affect a user is the MAXECSA value. The *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2*, SC32-1264, has a formula to calculate this value, or you can use a value of 400 and be safe. This value of 400 for MAXECSA is needed only for the Tracker started task (assuming that is the only writer), and the Controller could have a value of 0. Because suffix value F (for Tivoli Workload Scheduler for z/OS V8.2) is specified, EQQINITF loads module EQQSSCMF as in Example 1-1. In this example, TWSC is the Controller subsystem and TWST is the Tracker subsystem.

Example 1-1 IEFSSNxx subsystem table

```
SUBSYS SUBNAME(TWSC) INITRTN(EQQINITF) INITPARM ('0,F')
SUBSYS SUBNAME(TWST) INITRTN(EQQINITF) INITPARM ('400,F')
```

1.3.2 Updating the IEAAPFxx member

The Tivoli Workload Scheduler for z/OS modules in SEQQLMD0 that were copied to the linklist must also be APF (authorized program facility) authorized. To do so, enter the following entries into the IEAAPFxx member. (See Example 1-2 on page 8.) Enter the following example for the library that you have entered in the linklist in the next-to-last entry in the IEAAPFxx.

Important: If this library is moved, it will lose its authorization, and therefore should not be migrated.

Example 1-2 IEAAPFXX entry for authorization

```
TWS.LOADMODS VOL001,
```

1.3.3 Updating the SMFPRMxx member

You must make sure that the entries in the SMFPRMxx member contain the exits IEFUJI, IEFACTRT, and IEFU83, which are discussed in “SMF and JES exits installation” on page 11. We discuss how to configure these exits.

You also must make sure that the proper SMF records are being collected, as these exits depend on SMF records to update the events in the Tracker and the Controller.

These SMF records are needed:

- Type 14 records are required for non-VSAM data sets opened for INPUT or RDRBACK processing.
- Type 15 records are required for non-VSAM data sets opened for output.
- Type 64 records are required for VSAM data sets.
- Type 90 records support Daylight Saving Time automatically (optional).

To define the exits and records, the entries in Example 1-3 should be made in SMFPRMxx.

Example 1-3 Entries in SMFPRMxx to define the exits and records

```
SYS(TYPE(6,26,30),EXITS(IEFU83,IEFACTRT,IEFUJI))
SUBSYS(STC,EXITS(IEFUJI,IEFACTRT,IEFU83))
SUBSYS(JESn,EXITS(IEFUJI,IEFACTRT,IEFU83))
```

1.3.4 Updating the dump definitions

The sample JCL procedure for a Tivoli Workload Scheduler for z/OS address space includes a DD statement, and a dump data set is allocated by the EQQPCS02 JCL created by EQQJOBS. SYSMDUMP is the dump format preferred by the service organization.

Ensure that the dump options for SYSMDUMP (in SYS1.PARMLIB(IEADMPR00)) include RGN, LSQA, TRT, CSA, and GRSQ on systems where a Tivoli Workload Scheduler for z/OS address space will execute. To display the current SYSMDUMP options, issue the z/OS command DISPLAY DUMP,OPTIONS. You can use the CHNGDUMP command to alter the

SYSMDUMP options. This will only change the parameters until the next IPL is performed. The IEADMPR00 parameters are:

```
SDATA=(NUC,SQA,LSQA,SWA,TRT,RGN,SUM,CSA,GRSQ)
```

To dump a Tivoli Workload Scheduler for z/OS address space using the z/OS DUMP command, the SDUMP options should specify RGN, LSQA, TRT, CSA, and GRSQ. Consider defining these options as your system default.

Important: You must also make sure that the dump data sets are unique for each started task; otherwise the started task will not start.

1.3.5 Updating the XCF options (when using XCF)

Refer to Chapter 4, “Tivoli Workload Scheduler for z/OS communication” on page 87 to determine the method of communication to use. If possible, use XCF. As described in Chapter 3, XCF is much faster, and will improve performance.

Setting up XCF requires entries in the COUPLEnn member of Sys1. parmlib. Example 1-4 shows what could be configured for Tivoli Workload Scheduler.

Important: If XCF is used to connect the DataStore to the Controller, a specific XCF group must be defined that must be different from the one used to connect the Controller to the z/OS Tracker. These two separate XCF groups can use the same XCF transport class.

Example 1-4 Sys1.Pamlib entries for Tivoli Workload Scheduler

```
COUPLE SYSPLEX(PLEXV201) /* SYSPLEX name */
PCOUPLE(IM2.PLEXV201.CDS1,VOL001) /* Primary couple dataset */
ACOUPLE(IM2.PLEXV201.CDS2,VOL001) /* Alternate couple dataset*/
CLASSDEF CLASS(TCTWS) /* TWS transport class */
CCLASSLEN(152) /* Message length */
GROUP(TWSCGRP, TWSDS) /* TWSC group names */
MAXMSG(500) /* No of 1K message buffers*
```

The TWSCGRP parameter defines the Controller to Tracker Group, and the TWSDS defines the Controller to DataStore Group.

To set up the class definition as well as the group definition (for a temporary basis), you could use the command in Example 1-5.

Example 1-5 XCF command

```
SETXCF
START,CLASSDEF,CLASS=TCTWS,CLASSLEN=152,GROUP=(TWSCGRP,TWSDS),MAXMSG=50
0
```

1.3.6 VTAM parameters

If you are using VTAM as your connection between the Tracker/Controller and DataStore/Controller, you must update the Tivoli Workload Scheduler for z/OS parameter library and set up VTAM parameters. Example 1-6 lists parameters for the library. There are two separate LUs (logical units): one for the Controller/Tracker started tasks and one for the Controller/DataStore started tasks.

Note: These parameters are further explained in Chapter 5, “Initialization statements and parameters” on page 97.

Example 1-6 Parameters for one Controller, one Tracker, one DataStore

```
/*CONTROLLER PARAMETERS*/

OPCOPTS
NCFTASK(YES)
NCFAPPL(LU00C1T)
FLOPTS
CTLLUNAM(LU00C1D)
SNADEST(LU000T1.LU000D1,********.********)
ROUTOPTS SNA(LU000T1)

/*TRACKER PARAMETERS*/

OPCOPTS
NCFTASK(YES)
NCFAPPL (LU000T1)
TRROPTS
HOSTCON(SNA)
SNAHOST(LU00C1T)

/*Data Store PARAMETERS*/

DSTOPTS
HOSTCON(SNA)
DSTLUNAM(LU000D1)
CTLLUNAM(LU00C1D)
```

1.3.7 Updating the IKJTSOxx member

You must define the EQQMINOR module to TSO (time-sharing option) on each system where you install the scheduler dialogs. (This includes systems using a connection to the APPC Server.) Also, you must authorize the Tivoli Workload Scheduler for z/OS TSO commands on *every system where you install Tivoli Workload Scheduler*. If you do not authorize the Tivoli Workload Scheduler for z/OS TSO commands, they will work *only* on the system where the Controller is installed. Example 1-7 shows what might be configured on your system.

Example 1-7 IKJTSOxx parameters

```
AUTHTSF NAMES(IKJEFF76 IEBCOPY EQQMINOR)
AUTHCMD NAMES(BACKUP JSUACT OPINFO OPSTAT SRSTAT WSSTAT)
```

If present, IKJTSO00 is used automatically during IPL. A different IKJTSOxx member can be selected during IPL by specifying `IKJTSO=xx` for the IPL parameters. After the system is IPLed, the IKJTSOxx can be changed dynamically using the Set command:

```
T  IKJTSO=xx
```

1.3.8 Updating SCHEDxx member

To improve performance, you should define the Tracker and Controller address space as non-swappable. To do this, include the definition of the Tracker and Controller top load module, EQQMAJOR, in the program properties table (PPT) as not-swappable. To define the PPT, an entry in the SCHEDnn is required:

```
PPT PGMNAME(EQQMAJOR) NOSWAP
```

1.4 SMF and JES exits installation

The SMF and JES exits are the heart of tracking. These exits create events that the Tracker sends to the Controller so the current plan can be updated with the current status of the job being tracked.

Running EQQJOBS creates tailored sample members in the Install library that is used for output from EQQJOBS. These members are also located in the SEQQSAMP library as untailored versions.

If your z/OS system is a JES2 system, include these records in the JES2 initialization member JES2 Initialization Statements:

```
LOAD(OPCAXIT7)   /*Load TWS exit mod*/
EXIT(7) ROUTINES=OPCAENT7,STATUS=ENABLED /* Define EXIT7 entry point */
```

If your system is a JES3 system, activate the exits by linking them to a library that is concatenated ahead of SYS1.JES3LIB. Alternatively, you can replace the existing exits in SYS1.JES3LIB with the Tivoli Workload Scheduler–supplied IATUX19 and IATUX29 exits. For more information, refer to *z/OS JES3 Initialization and Tuning Reference*, SA22-7550. If you get RC=4 and the warning `ASMA303W Multiple address resolutions may result when you assemble IATUX19 running the EQQJES3/EQQJES3U sample`, you can ignore the message. If Version IEV90 of the compiler reports errors, remove the RMODE=ANY statement from the sample exit.

Table 1-2 shows the Tivoli Workload Scheduler for z/OS exits and their functions.

Table 1-2 Exits and their functions

Exit name	Exit type	Sample exit	Sample JCL/usermod	Event supported	Event type
IEFACTRT	SMF	EQQACTR1	EQQSMF	Job and step completion	3J,3S
IEFUJI	SMF	EQQUJI1	EQQSMF	Job start	2
IEFU83	SMF	EQQU831	EQQSMF	End of print group and purge, and dataset triggering support	4,5,S
EXIT7	JES2	EQQX74	EQQJES2 EQQJES2U	JCT I/O exit for JES2	1,3P
IATUX19	JES3	EQQX191	EQQJES3 EQQJES3U	Output processing complete	3P
IATUX20	JES3	EQQX201	EQQJES3 EQQJES3U	On the JobQueue	1

1.5 Running EQQJOBS

EQQJOBS is a CLIST/ISPF dialog that is supplied in SYSx.SEQQCLIB. It can tailor a set of members to:

- Allocate data sets
- Build a customized set of parms
- Customize the procedures for the started task
- Create long-term plan and current plan
- JES/SMF exit installation

1.5.1 How to run EQQJOBS

You must first create two data sets for output, one for the Skeleton JCL and one for the Installation JCL. One suggestion for a name is HLQ.SKELETON, HLQ.INSTALL.JCL. Note that this naming suggestion is using full words such as SKELETON, INTSTALL, and JCL instead of abbreviations as described in the *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264 (instljcl,jclskels). In the same manual, note the recommendation to put the DataStore JCL into the HLQ.INSTALL.JCL instead of a separate library (instds). This will keep all the install JCL together in one data set. This is discretionary and an effort to simplify the recognition of data set names. These libraries should be FB, LRECL 80, and a PDS (partitioned data set). See Example 1-8.

Example 1-8 Pre-allocation of EQQJOBS data sets

```
//ALLOC   JOB ,,CLASS=A
/*JOBPARM SYSAFF=SC64
//*
//STEP1   EXEC PGM=IEFBR14
//EQQSKL  DD   DSN=TWS.SKELETON,DISP=(,CATLG),
//        DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000),UNIT=3390,
//        SPACE=(CYL,(5,2,10))
//EQQJCL  DD   DSN=TWS.INSTALL.JCL,DISP=(,CATLG),
//        DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000),UNIT=3390,
//        SPACE=(CYL,(5,2,10))
```

To run the EQQJOBS CLIST, you can use the REXX executable in Example 1-9 to allocate the necessary libraries and invoke the EQQJOBS CLIST.

Example 1-9 REXX exec to run EQQJOBS CLIST

```
/*REXX*/
"ALTLIB ACT APPL(CLIST) DSN('SYSx.SEQQCLIB') UNCOND"
address ISPEXEC
"LIBDEF ISPPLIB DATASET ID('SYSx.SEQQPNL0')"
"LIBDEF ISPTLIB DATASET ID('SYSx.SEQQTBL0')"
"LIBDEF ISPMLIB DATASET ID('SYSx.SEQQMSG0')"
"LIBDEF ISPSLIB DATASET ID('SYSx.SEQQSKL0',
'SYSx.SEQQSAMP')"
address TSO
"EQQJOBS"
Address "TSO" "ALTLIB DEACTIVATE USER(CLIST)"
Address "TSO" "FREE F(SYSUPROC)"
"LIBDEF ISPPLIB DATASET ID('SYSx.SEQQPNL0')"
"LIBDEF ISPTLIB DATASET ID('SYSx.SEQQTBL0')"
```

```
"LIBDEF ISPMLIB DATASET ID('SYSx.SEQQMSG0')"
"LIBDEF ISPSLIB DATASET ID('SYSx.SEQQSKL0',
'SYSx.SEQQSAMP')"
exit
```

1.5.2 Option 1

When you run the EQQJOBS CLIST, you see the options shown in Figure 1-1.

1. Select option 1 to begin.

Note: Entering PF1 gives an explanation of each field on EQQJOBS panel.

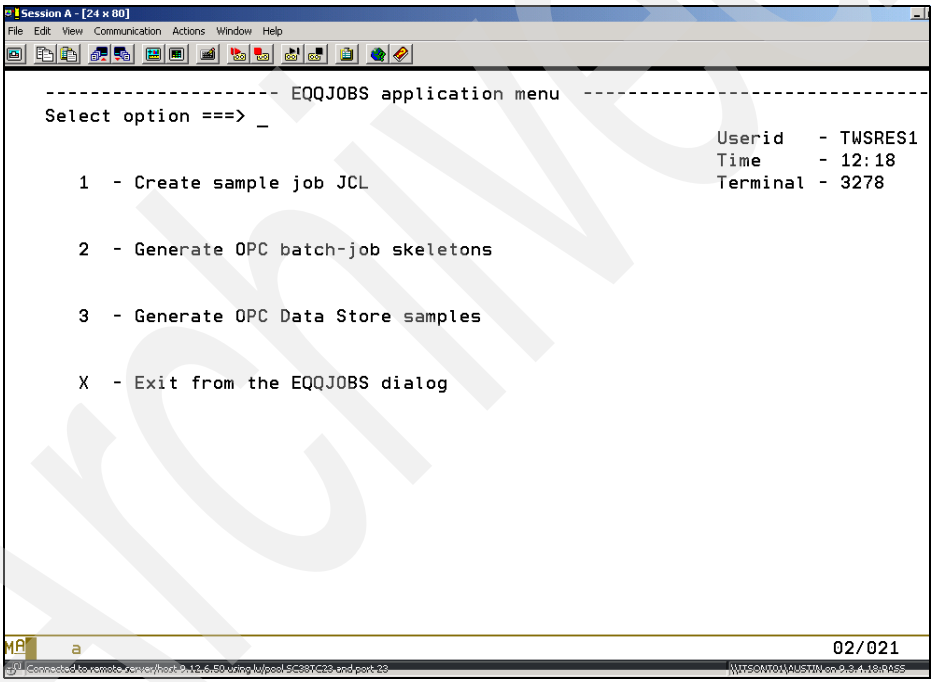


Figure 1-1 EQQJOBS primary menu

2. After entering the first option, make the entries shown in Figure 1-2. HLQ is the name you will use for all data sets during the install process. HLQ.INSTALL.JCL *must be the data set that you pre-allocated prior to running EQQJOBS.* SEQQMSG0 is the library created by the SMP/E install.

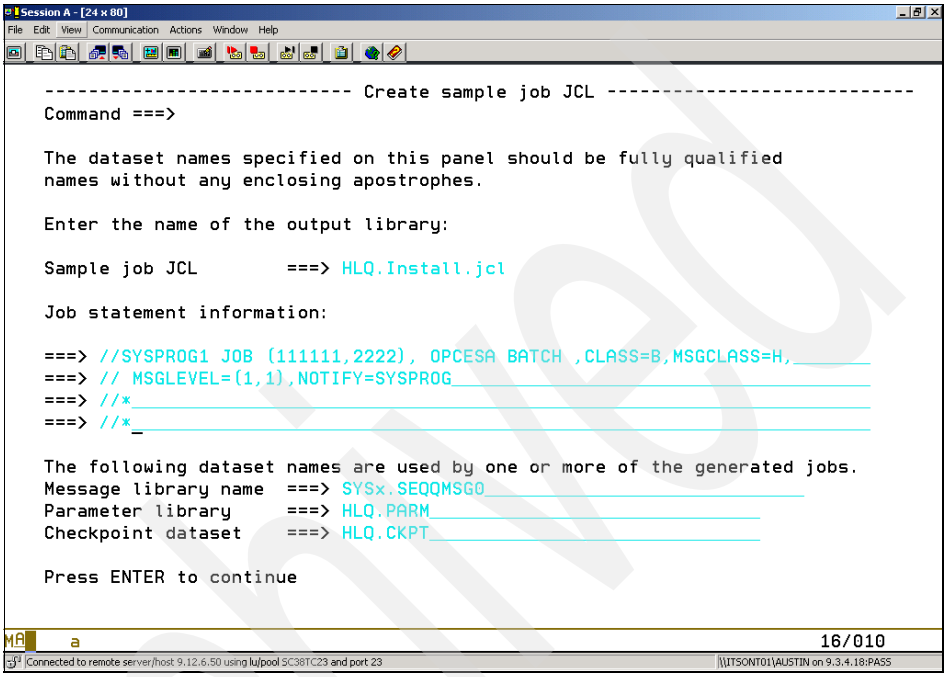


Figure 1-2 EQQJOBS entries for creating JCL

3. Press Enter to get the next set of options needed for EQQJOBS, carefully noting the names of the data sets.

> **Note:** Some installations require a difference in naming convention between VSAM and non-VSAM.

This step sets up the HLQ names for all data sets that will be created for the started task jobs (Figure 1-3).

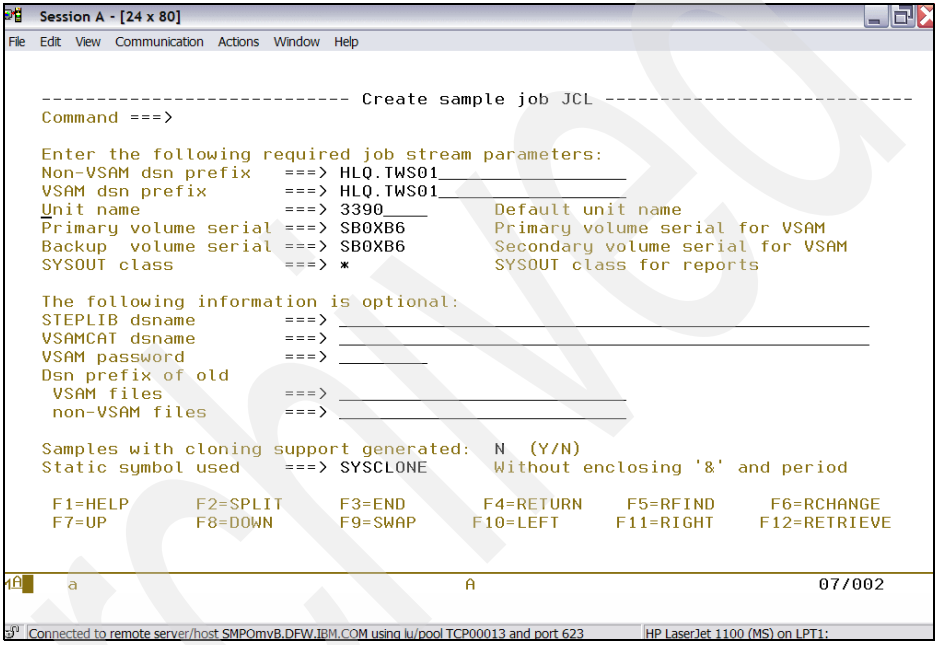


Figure 1-3 Data set naming entries

4. Press Enter to display the window in Figure 1-4. On this frame we will not install the end-to-end feature.

   Pay special attention to the Reserved Destination, as this is the setup for the DataStore/Controller parameter for JES control cards. Also, END TO END FEATURE should be **N**, unless you are installing that particular feature.

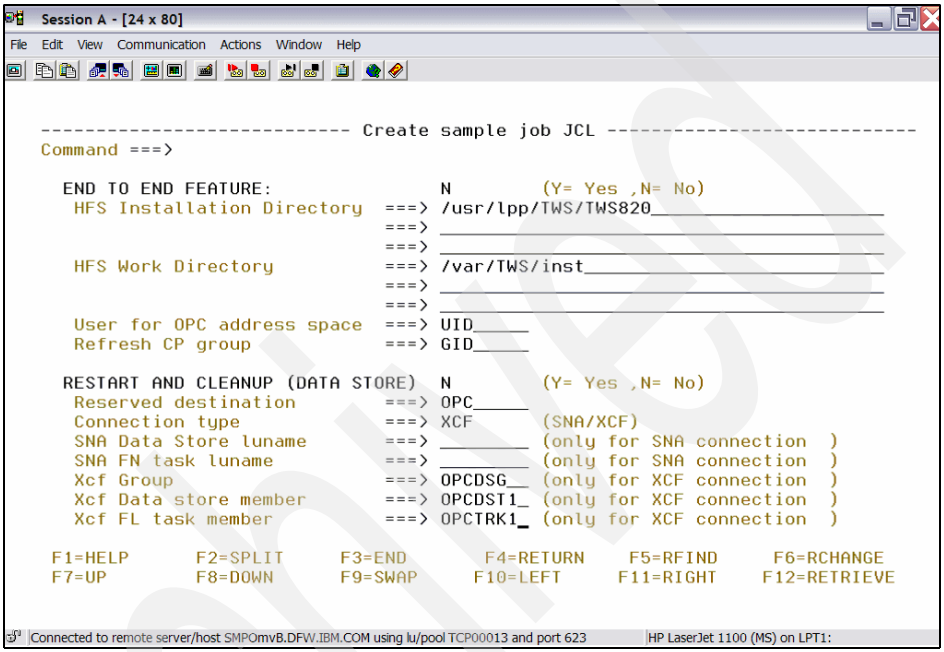


Figure 1-4 EQQJOBS data set entries

5. After you press Enter, EQQJOBS will display messages showing the members that it has created. Table 1-3 shows the members and gives a short description of each. Most members are self-documenting and contain comments that are self-explanatory. The install will not necessarily use all members.

Table 1-3 Install members

Member	Description
EQQCONOP	Sample parameters for the Controller
EQQCONO	Sample started task procedure for the Controller
EQQCONP	Sample parms for Controller/Tracker in the same address space
EQQCON	Sample started task procedure for Controller and Tracker in same address space

Member	Description
EQQDPCOP	JCL and usage notes for copy VSAM functions
EQQE2EP	Sample parms for E2E
EQQICNVH	Sample jobs to migrate history DB2 tables
EQQICNVS	Migrates VSAM files
EQQJES2	Assembles and link-edits Jes2 exit7
EQQJES2U	Installs the JES2 usermod
EQQJES3	Assembles and link-edits a JES3 exit
EQQJES3U	Installs the JES3 usermod
EQQRST	Resets the USS environment for E2E
EQQPCS01	Allocates unique data sets within the sysplex
EQQPCS02	Allocates non-unique data sets
EQQPC03	Allocates VSAM copy data sets
EQQPCS05	Allocates files used by a Controller for E2E
EQQPCS06	Allocates VSAM data sets for E2E
EQQPCS07	Allocates VSAM data sets for Restart and Cleanup
EQQSAMPI	Copies sample databases from the sample library to VSAM data sets
EQQSERP	Sample initial parameters for a Server
EQQSER	Sample started task procedure for a Server
EQQSMF	Updates SMF exits for Tivoli Workload Scheduler
EQQTRA	Sample started task procedure for a Tracker
EQQTRAP	Sample initial parameters for a Tracker

This completes Option 1. Now proceed to Option 2.

1.5.3 Option 2

Option 2 of EQQJOBS generates the members in the Skeleton JCL data set.

1. Select option 2 on the main panel and enter the parameters in Figure 1-5. This step builds the ISPF skeletons necessary for Tivoli Workload Scheduler for z/OS to do such things as build the long-term plan or 'current plan, set up the audit function batch job, and build jobs to run the reports. These skeleton JCL members should be analyzed to determine whether the space for the long-term planning and current planning data sets are adequate.

   After running EQQJOBS it would be helpful to expand the size of the sort data sets, as well as the temporary data sets if the database is large.

   Press Enter.

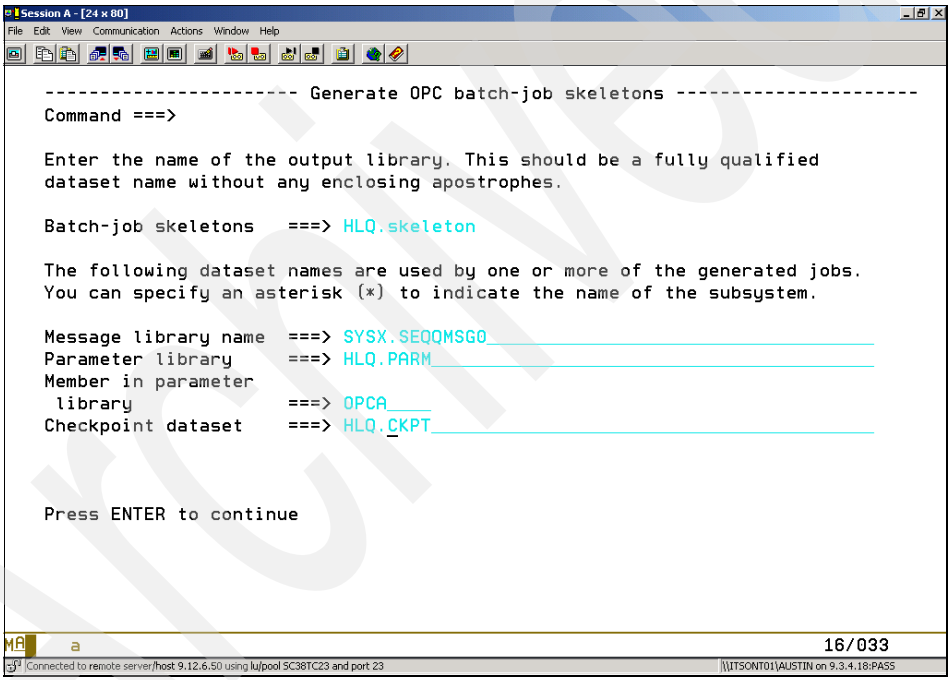


Figure 1-5 EQQJOBS generate skeletons

2. When entering the Checkpoint and Parameter data sets (Figure 1-6), note that the JCL to create this data set was created in Option 1. You *should* use the same name to refer to members, EQQPCS01 (in the install data set).

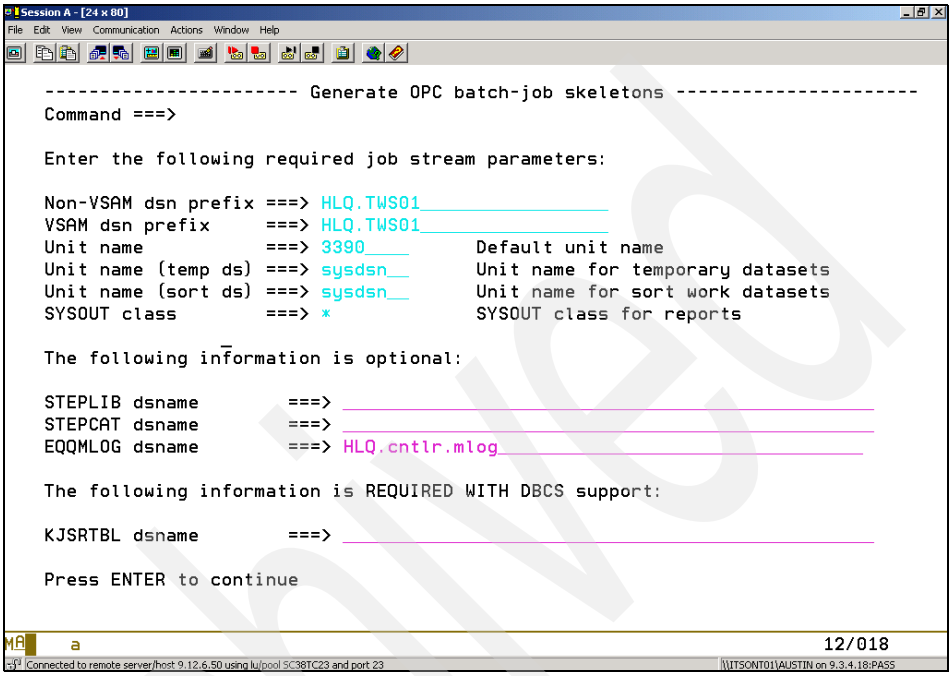


Figure 1-6 Generate skeletons

3. Press Enter to display the window in Figure 1-7). Make sure that you set RESTART AND CLEAN UP to Y if you will use DataStore and do job restarts.

   Specify the name of the data set in which DP Extend and Replan writes tracklog events with the DD EQQTROUT. (Without this tracklog you will have no history for the Audit Function to run against.) Entry EQQTROUT is optional but recommended. Leave blank if you want the corresponding DD card for these jobs to specify DUMMY.

   Fill out EQQAUDIT for a default report name.

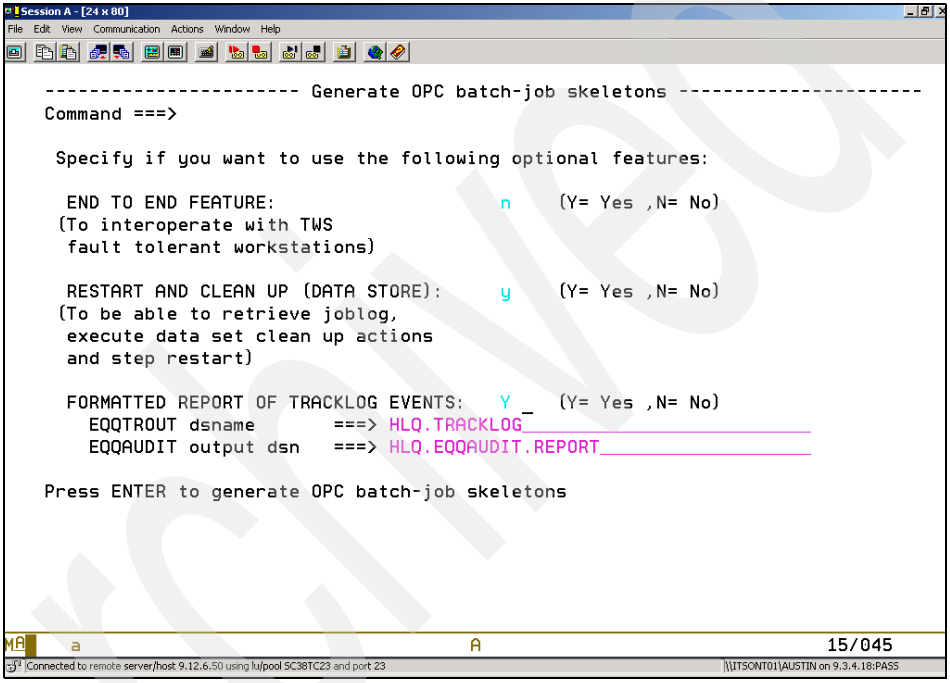


Figure 1-7 Generate skeleton JCL

> **Important:** Make sure that the EQQAUDNS member is reviewed, modified, and put into a Procedure library because otherwise Tivoli Workload Scheduler for z/OS Audit will not work. An example in Appendix B, “EQQAUDNS member example” on page 673 shows the EQQAUDNS member that resides in the HLQ.SKELETON DATASET (output from EQQJOBS). This member has a comment of /* <<<<<<< */ to indicate that a review of the data set name is necessary.

Table 1-4 on page 22 shows what members were created in the Skeleton Library. Note that the daily and long-term planning should have the Temporary

and Sort data sets increased in size; otherwise you risk abends during production.

Table 1-4 Skeleton Library members

Member	Description
EQQADCOS	Calculate and print run dates of an application
EQQADDES	Application cross-reference of external dependencies
EQQADPRS	Application print program
EQQADXRS	Application cross-reference program
EQQADX1S	Application cross-reference of selected fields
EQQAMUPS	Application description mass update
EQQAPARS	Procedure to gather diagnostic information
EQQAUDIS	Extract and format job tracking events
EQQAUDNS	Extract and format job tracking events (ISPF invocation)
EQQDPEXS	Daily planning next period
EQQDPPRS	Daily planning print current period results
EQQDPRCS	Daily planning replan current period
EQQDPSJS	Daily planning DBCS sort step
EQQDPSTS	Daily planning normal sort step
EQQDPTRS	Daily planning plan a trial period
EQQJVPRS	Print JCL variable tables
EQQLEXTS	Long-term planning extend the long-term plan
EQQLMOAS	Long-term planning modify all occurrences
EQQLMOOS	Long-term planning modify one occurrence
EQQLPRAS	Long-term planning print all occurrences
EQQLPRTS	Long-term planning print one occurrence
EQQLTRES	Long-term planning create the long-term plan
EQQLTRYS	Long-term planning trial
EQQOIBAS	Operator instructions batch program
EQQOIBLS	Operator instructions batch input form a sequential data set

Member	Description
EQQSSRES	Daily planning Symphony Renew
EQQTPRPS	Print periods
EQQTPRTS	Print calendars
EQQWMIGS	Tracker agent jobs migration program
EQQWPRTS	Print workstation description

1.5.4 Option 3

DataStore is an optional started task, but it is needed to do Restart/CleanUp, as well as viewing sysouts from the ISPF panels. Therefore, it should be included in the installation.

1. From the main EQQJOBS primary window, enter 3 as an option.
2. This opens the window in Figure 1-8, which is the beginning of the building of the DataStore data set allocation JCL and parameters. Enter the information shown and press Enter.

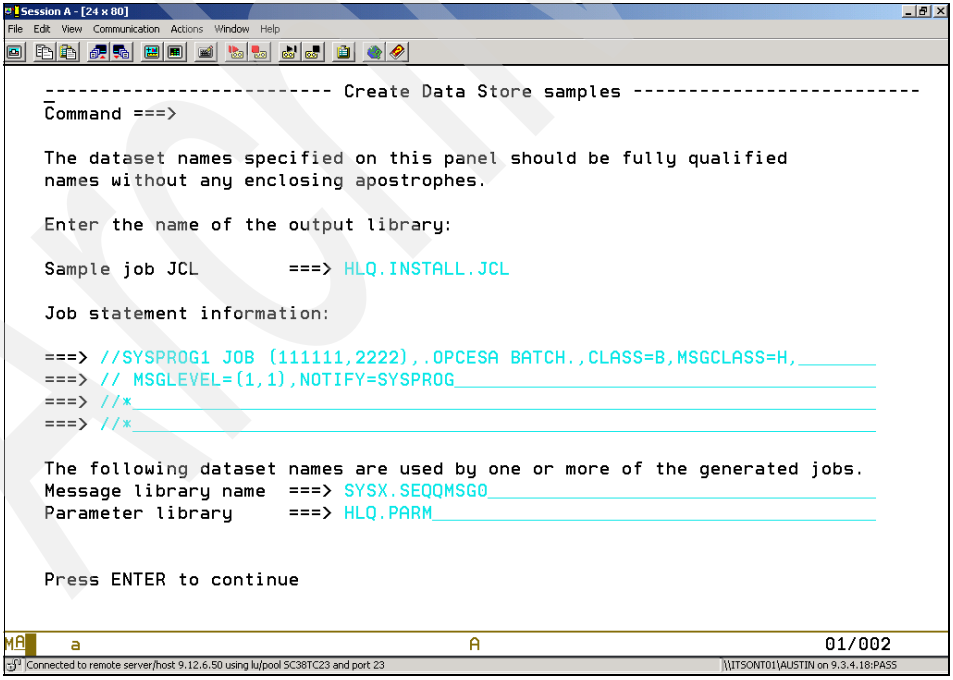


Figure 1-8 Generate DataStore samples

3. Enter the VSAM and Non-VSAM data set HLQs (Figure 1-9), and press Enter.

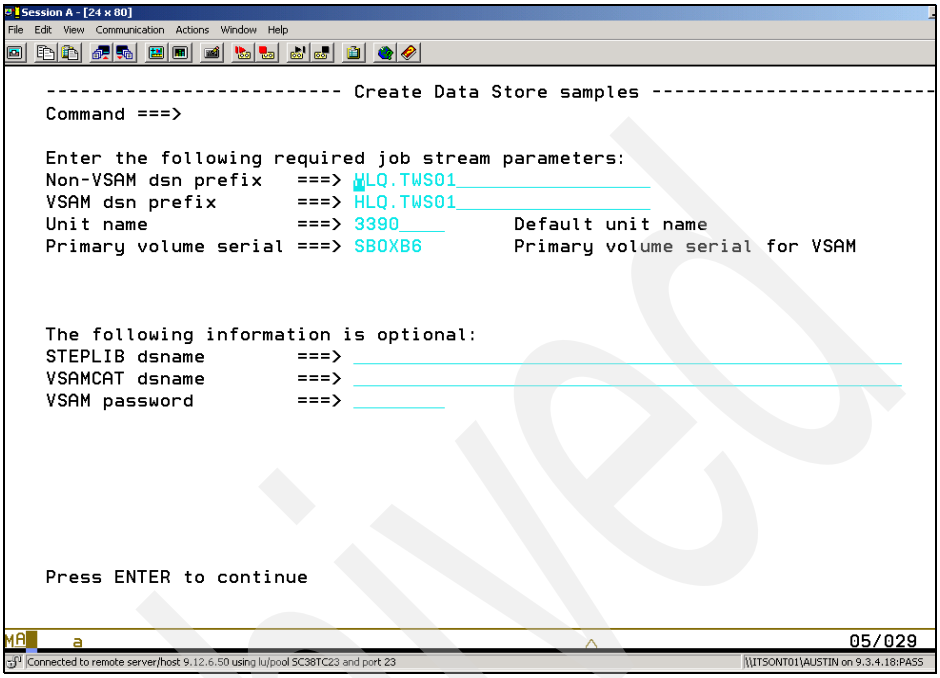


Figure 1-9 Create DataStore samples

4. This displays the window in Figure 1-10. If you are using XCF, use XCF for Connection type, and enter the XCF group name, a member name, FLtaskname, and other fields. For further explanation of these parameters, refer to Chapter 3, “The started tasks” on page 69 and *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265.

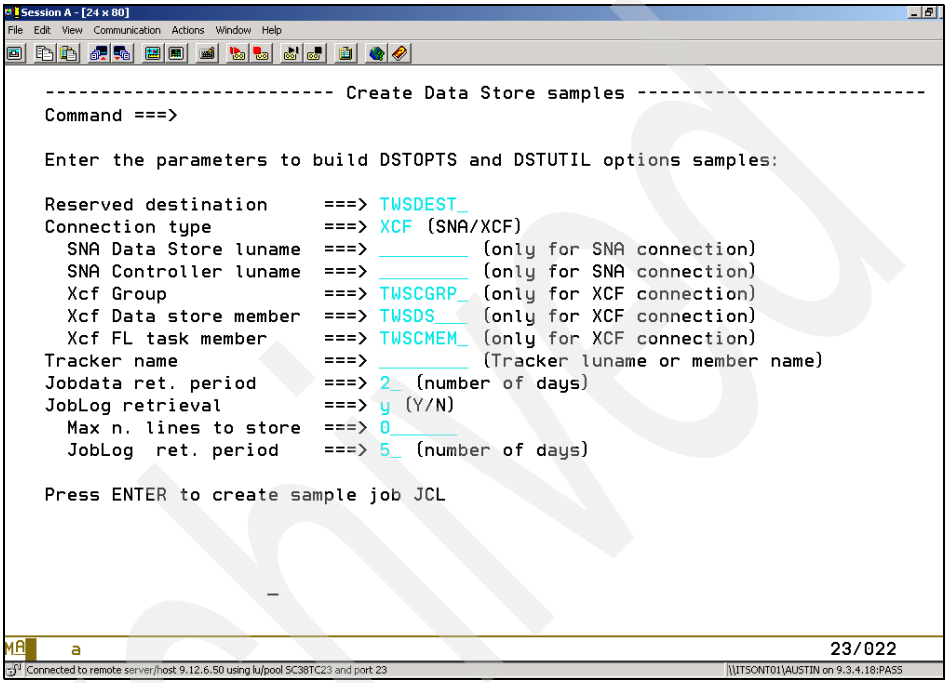


Figure 1-10 Create DataStore samples

5. Press Enter, and EQQJOBS creates new members in the install data set and completes the EQQJOBS step. The members shown in Table 1-5 are created.

Table 1-5 Members created in Option 3

Member	Description
EQQCLEAN	Sample procedure invoking EQQCLEAN program
EQQDSCL	Batch cleanup sample
EQQDSCLP	Batch cleanup sample parameters
EQQDSEX	Batch export sample
EQQDEXP	Batch export sample parameters
EQQDSIM	Batch import sample

Member	Description
EQQDSIMP	Batch import sample parms
EQQDSRG	Batch sample reorg
DQQDSRI	Batch recovery index
EQQDSRIP	Batch recovery index parameters
EQQDST	Sample procedure to start DataStore
EQQDSTP	Parameters for sample procedure to start DataStore
EQQPCS04	Allocate VSAM data sets for DataStore

1.6 Security

Chapter 7, “Tivoli Workload Scheduler for z/OS security” on page 163 discusses security topics in detail. We recommend that you read this chapter and understand the security considerations for Tivoli Workload Scheduler for z/OS before doing the installation. Before you start the Controller, Tracker, or DataStore, you must authorize the started tasks; otherwise the started task will get RACF® errors when you attempt to start it.

Important: If you are getting errors and suspect that you have an RACF error, check the syslog for messages beginning with `ICH`.

Next, authorize Tivoli Workload Scheduler for z/OS to issue JES (job entry subsystem) commands and to give authority to access the JES Spool. If there is a problem submitting jobs and an RACF message appears, you might suspect that one of the Tivoli Workload Scheduler/JES authorizations is not setup properly.

You must make a decision if you want to allow the Tivoli Workload Scheduler for z/OS Tracker to submit jobs using surrogate authority. Surrogate authority is allowing one user ID (the Tracker if you so choose) to submit work on behalf of another user ID. Giving the Tracker surrogate authority enables it to submit jobs with the Tracker’s user ID. If you choose not to do this, you should use EQQUX001 exit and submit jobs with the ruser user ID. Using the ruser user ID enables Tivoli Workload Scheduler for z/OS to submit the job with the ID that the exit is providing. This does require coding the exit and making a decision about how the user ID gets added on the submit (see 7.2, “UserID on job submission” on page 165 for more detail about how to use the ruser user ID.) Different levels of authority are required for users with different job functions (such as

schedulers, operators, analysts, and system programmers). An RACF group profile must be set up for each of these groups. Chapter 7, “Tivoli Workload Scheduler for z/OS security” on page 163 has examples of each of these groups and how you might set them up.

The *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264, and *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265, cover security in detail.

1.7 Allocating the data sets

EQQJOBS has made allocating the data sets much easier for you. EQQJOBS creates multiple tailored members in the install library. If these are massively wrong, you may want to rerun EQQJOBS.

Note: EQQJOBS can be rerun as many times as you wish.

You must inspect the members you are going to use to make sure that each is set up properly before running.

Important: Check for size, names of the data sets, and your DFSMS convention issues (for example, VOLSER).

We have included a spreadsheet with this book to help with sizing. The spreadsheet can be downloaded from the ITSO Web site. For download instructions, refer to Appendix C, “Additional material” on page 679.

For more information about the sizing of the data sets, refer to *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264, where you will find some methods to calculate the data sets.

Run EQQPCS01, which was created previously with EQQJOBS in the Install library, to allocate the unique data sets shown in Table 1-6.

Table 1-6 EQQPCS01 data sets allocated

DDNAME	Description
EQQADDS	Application description file
EQQWSDS	Workstation, calendar, period description
EQQRDDS	Special Resource definitions
EQQLTDS	Long-term plan

EQQLTBKP	Long-term plan backup
EQQLDDS	Long-term plan work file
EQQNCPDS	New current plan
EQQCP1DS	Current plan one
EQQCP2DS	Current plan two
EQQCXDS	Current plan extension
EQQNCXDS	New current plan extension
EQQJS1DS	JCL repository one
EQQJS2DS	JCL repository two
EQQSIDS	Side information, ETT configuration file
EQQOIDS	Operation instruction file

Run EQQPCS02 after analyzing and making the necessary modifications. You should create an MLOG data set for each started task and a unique EV01/02 for each Tracker and each Controller. You should also make unique dump data sets for each started task. This job creates the data sets shown in Table 1-7**.**

Note: Each started task requires its own dump data set and that the Tracker and the Controller each have their own EV data sets.

Table 1-7 EQQPCS02 allocated data sets

DDNAME	**Description**
EQQEV01/02	Event data sets
EQQINCWRK	JCC Incident work file
EQQDUMP	Dump data set
EQQMDUMP	Dump data set
EQQMLOG	Message logging data set
EQQTROUT	Input to EQQAUDIT
AUDITPRINT	EQQAUDIT output file

To create the DataStore data sets, run EQQPCS04 and EQQPCS07. The size of these data sets is entirely dependent on how large the sysouts are that you are storing in DataStore, and how long you leave them in DataStore before cleaning

them up with the DataStore cleanup job. One thing to keep in mind is that you may add data sets to the DataStore if you are running out of space. Table 1-8 shows the DataStore files that will be allocated by EQQPCS04 and EZZPCS07.

Table 1-8 DD statements for DataStore

DDNAME	Description
EQQPKxx	Primary index files
EQQSDFxx	Structured data files
EQQSKIxx	Secondary index file
EQQUDFxx	Unstructured data files

1.7.1 Sizing the data sets

To size DataStore data sets you may use Table 1-9 for the VSAM files, or use the spreadsheet that you can download from ITSO Web site as a guideline to size data sets. (See Appendix C, “Additional material” on page 679 for download instructions.)

As a base, calculate a figure for all your jobs and started tasks that are controlled by Tivoli Workload Scheduler. Add to this figure the expected space required for jobs and started tasks in the current plan (Example 1-9).

Table 1-9 VSAM data set size calculation

Data set	Number of	Multiplied by
Application description (EQQADDS)	Application and group definitions Run cycles Positive run days Negative run days Operations Internal dependencies External dependencies Special resources Operation Extended Information Variable tables Variables Variable dependencies Extended Name	208 120 3 3 110 16 84 64 200 98 476 88

Data set	Number of	Multiplied by
Current plan (EQQCPnDS)	Header record (one only) Workstations Workstation open intervals Workstation access method data Occurrences Operations Dependencies Special resource references Operation Extended Information Jobs Executed steps Print operations Unique application names Operations currently in error Reruns of an operation Potential predecessor occurrences Potential successor occurrences Operations for which job log information has been collected Stand alone clean up Restart and clean up operinfo retrieved Number of occurrences	188 212 48 72 302 356 14 64 200 116 20 20 64 264 264 32 24 111 70 44 43
JCL repository (EQQJSnDS)	Number of jobs and started tasks Total lines of JCL Operations for which job log information has been collected Total lines of job log information	80 80 107 143
Long-term plan (EQQLTDS)	Header record (one only) Occurrences External dependencies Operations changed in the LTP dialog	92 160 35 58
Operator instruction (EQQOIDS)	Instructions Instruction lines	78 72
Special resource database (EQQRDDS)	Resource definitions Defined intervals Entries in the WS connect table	216 48 8
Side information file (EQQSIDS)	ETT requests	128

Data set	Number of	Multiplied by
Workstation/calendar (EQQWSDS)	Calendars Calendar dates Periods Period origin dates Workstation closed dates Workstations Workstation access method data Interval dates Intervals	96 52 94 6 80 124 72 52 32

Note: Use the preceding table for the following items:

1. Use the current plan data set calculation (EQQCPnDS) for the new current plan data sets (EQQNCPDS and EQQSCPDS).
2. Use the long-term-plan data set calculation (EQQLTDS) for the long-term-plan work data set (EQQLDDS) and the long-term-plan backup (EQQLTBKP).
3. Use the special resource database calculation (EQQRDDS) for the current plan extension data set (EQQCXDS) and the new current plan extension (EQQNCXDS).

Non-VSAM sizing

Most non-VSAM data sets created by EQQJOBS are usable, with the exception of a few:

- EQQJBLIB

  The size of this library depends on the number of members you intend to put into it. Remember that the directory blocks have to be increased as the number of members grows. Also in the Started Task for the Controller you may want to concatenate your JCL libraries to the DD instead of copy your members into this library.

- EQQEVxx

  A good starting point for this data set is 25 cylinders as it is a wrapping data set and this will give you enough space for a long period before the data set wraps. The installation guide has some specifics about size formulas.

- EQQMLOG

  A good starting point is 25 cylinders, but remember that the bigger you make it the longer it takes to search to the end. If the data set is too big when you search, your screen could be locked for a good period of time.

DataStore sizing

DataStore VSAM data files consist of:

- Data files for structured and unstructured data
- Primary index
- Secondary index

Data files

The DataStore distinguishes VSAM data file (DD) types by their names:

- Structured DDs are called EQQSDFnn
- Unstructured DDs are called EQQUDFnn

Although the data file structure for these two types is the same, their content and purpose differ, as described below.

Unstructured data files

The unstructured files are needed to be able to fetch the sysouts from DataStore. The unstructured data files contain the SYSOUTs in a flat form, as provided by the JES spool. You can check the SYSOUT with the BROWSE JOBLOG function. Note that if requested to, the unstructured data file also can store the user SYSOUTs (which can utilize large amounts of DASD). The activation of the unstructured data files is optional, depending on appropriate DataStore parameters.

Within an unstructured data file, every SYSOUT, consisting of n logical records, takes at least one page of data (4096 bytes). The size of the VSAM data file depends on the following factors:

- The typical size of the SYSOUT for jobs that have to be stored (also consider the MAXSTOL parameter that specifies the number of user SYSOUT lines to be stored).
- The average number of jobs that run every day.
- The retention period of job logs in DataStore.
- The number of data files that you want to create (from 1 to 99). You can calculate the number of pages that you need in this way:
  - Calculate the maximum number of job logs that can be stored at a given time. To do this, multiply the number of jobs running in a day by the number of days that you want the job logs to be available.
  - Calculate the average number of pages that are needed for every job log. This depends on the average number of lines in every SYSOUT and on the average SYSOUT line length. At least one page is needed for every job log.

– Calculate the total number of required pages by multiplying the number of job logs stored concurrently by the average number of pages for every SYSOUT.
– Calculate the number of pages required for each file by dividing the previous result by the number of data files you want to create.
– Determine size of each data file according to the media type and space unit for your installation.

This is an example of calculating for unstructured data files:

A company runs 1,000 jobs every day on a single system, and each job generates around 4,000 lines of SYSOUT data. Most lines are 80 characters long. Restart and Cleanup actions are taken almost immediately if a job fails, so it is not necessary to keep records in the DataStore for more than one day. A decision is made to spread the data over 10 files. The maximum number of logs stored at a given time is: 1,000 * 1 = 1000. As each log is about 4,000 lines long, and each line is about 80 characters long, the number of bytes of space required for each is: 4,000 * 80 = 320,000. Thus, the total number of bytes of space required is: 320,000 * 1000 = 320,000,000. If four files were used, each file would hold the following number of bytes of data: 320,000,000 / 4 = 80,000,000. If 3390 DASD was used, each file would require this number of tracks: 80,000,000 / 56,664 = 1,412 or this number of cylinders: 80,000,000 / 849,960 = 94.

Structured data files

The structured data files contain job log SYSOUTs in a form based on the parsing of the three components of the job log: the JESJCL, the JESYSMSG, and the JESMSGLG (especially the first two). User SYSOUTS are excluded from the structuring mode. Each stored job log consists of two distinct parts:

- A number of pages, each consisting of 4096 bytes dedicated to the expanded JCL
- A number of pages dedicated to a complete, hierarchically ordered set of structured elements for the Restart and Cleanup functions

Therefore, the minimum page number used by a structured SYSOUT is 2, and the medium space usage depends on the job complexity. To determine the optimal dimension for the structured data files, follow the instructions provided for the allocation of the unstructured data file, but take into account that the user SYSOUTs are not present. For the medium structured SYSOUTs, apply the criteria used for the unstructured job log: The larger memory requirement of the small, structured SYSOUTs, compared to the corresponding unstructured form, is balanced by the larger memory requirement of the unstructured form when the SYSOUT complexity increases.

Primary index

Every user SYSOUT data set requires one row. The three z/OS SYSOUT data sets together require one row. Every row in the index has a fixed 77-character length. To set the right size of VSAM primary index file, multiply the average number of SYSOUT data sets per job by the maximum number of jobs stored concurrently in the database. This value is the maximum number of rows in the primary index — it should be increased by an adequate margin to cope with peaks in your workload and to allow for growth.

To find the total space quantity to allocate for VSAM primary index, you should multiply this adjusted maximum row number by the total length of the record. For example:

The vast majority of the 1000 jobs run daily by the same company of the previous example generates a single user SYSOUT data set, along with the usual system data sets. Thus, the maximum number of rows in the index is: 2 * 1,000 = 2,000. Allowing 50% for growth, the space required for the index is: 3000 * 77 = 231,000 bytes. On a 3390 this is 231,000 / 56664 = 4 tracks.

Secondary index

The secondary index is a variable-length key-sequenced data set (KSDS). Because it can be a single record that corresponds to a specific secondary-key value, it can trace many primary keys. Currently, a secondary key value is associated to a single primary key only, and, for this reason, each SYSOUT in the secondary index requires one row of 76 characters.

To set the size of the VSAM secondary index file, perform the following steps:

- Multiply the average number of SYSOUT data sets for each job by the maximum number of jobs stored currently in the database. The result is the maximum number of rows in the secondary index.
- Increase this value to cope with peaks in workload and to allow for growth.
- Multiply this adjusted value by the total length of the record. This gives the total space for allocating for the VSAM secondary index.

Characteristics of the local DataStore

Local store data sets are the DataStore data sets that are used by the Controller, and contain SYSOUTs that will be needed for restart.The criteria for setting the size of the VSAM local DataStore differ from those for the main DataStore. Therefore, note the following items:

- Only those SYSOUTs in the main DataStore that are subject to Restart and Cleanup are also stored in the local DataStore.
- Because unstructured data is not subject to Restart and Cleanup, the local DataStore requires significantly less space.

1.8 Creating the started tasks

The EQQJOBS created the started task procedures in the INSTALL library. The member EQQCONO is the procedure for the Controller, EQQDST is the procedure for the DataStore, EQQSER is the procedure for the server, and EQQTRA is the procedure for the Tracker. After reviewing or modifying these procedures, you should move them to a production procedure library.

It is recommended that the Tracker be in *the same performance group as JES*, and the Controller be in a high-performance group. The Tracker/Controller must made not-swappable, so they will maintain their address space in storage; otherwise there will be a severe performance degradation. (See 1.3.8, “Updating SCHEDxx member” on page 11.)

When starting Tivoli Workload Scheduler, you should always start the Tracker first, followed by the Controller, the DataStore, then a server, if required. The reverse order should be used on bringing Tivoli Workload Scheduler for z/OS down. Note that when getting ready to bring the system down the Tracker should be the last task to go down before JES.

Important: Tivoli Workload Scheduler for z/OS should never be cancelled, only stopped because the database could be compromised if using the cancel command.

See Chapter 3, “The started tasks” on page 69 for more details.

1.9 Defining Tivoli Workload Scheduler for z/OS parameters

For defining Tivoli Workload Scheduler for z/OS parameters, refer to Chapter 5, “Initialization statements and parameters” on page 97.

1.10 Setting up the ISPF environment

Follow these steps to set up the ISPF environment:

1. Allocate the SEQQTBL0 library (Example 1-10) to the ISPFTLIB.

Example 1-10 ISPF tables

```
EQQACMDS ISPF command table
EQQAEDIT Default ISPF edit profile
```

```
EQQELDEF Default ended-in-error-list layouts
EQQEVERT Ended-in-error-list variable-entity read table
EQQLUDEF Default dialog connect table
EQQRLDEF Default ready-list layouts
EQQXVART Dialog field definitions
```

Table EQQLUDEF contains values used when establishing the connection between the scheduler dialog user and the Controller. These default values are set initially for your installation by the system programmer. Individual users can then modify the values to suit their requirements.

2. Modify the table, adding the following information:
   - The names of the Controllers in your installation.
   - When a Controller is accessed remotely, the combination of the Controller name and the LU name of a server set up to communicate with it.
   - The set of dialog–Controller connections that are to be available to all dialog users.
3. Allocate the SEQQCLIB to the TSO logon procedure SYSPROC. You can use the REXX exec shown in Example 1-11 to access the ISPF dialog.

Example 1-11 REXX to invoke Tivoli Workload Scheduler for z/OS dialog

```
/*REXX*/
Address ISPEXEC
    ISPEXEC "CONTROL ERRORS RETURN"
    ISPEXEC "LIBDEF ISPPLIB DATASET ID(‘SYSxSEQQPENU')“
    ISPEXEC "LIBDEF ISPMLIB DATASET ID('SYSx.SEQQMSGO')"
    ISPEXEC "LIBDEF ISPTLIB DATASET ID('SYSx.SEQQTBLO')“
    ISPEXEC "LIBDEF ISPSLIB DATASET ID(‘HLQ.SKELETON')"
ISPEXEC "SELECT PANEL(EQQOPCAP) NEWAPPL(TWSC) PASSLIB"
exit
```

4. You can modify the primary panel of ISPF to access the Tivoli Workload Scheduler for z/OS dialog. Example 1-12 shows how to make that modification.

Example 1-12 ISPF primary panel modifications

```
)BODY ...
1 ....... - ............
2 ....... - ............
. ....... - ............
O TWS - Tivoli Workload Scheduler for z/OS <<<<<<<<<< Modify this
)PROC ...
.......
```

```
2 , ....
. , ....
0 , 'PANEL(EQQOPCAP) NEWAPPL(EQQA)' <<<<<<<<< Modify this
. , .... ...
)END
```

1.11 Configuring Tivoli Workload Scheduler for z/OS; building a current plan

Before running long-term and current plans and submitting a job, you must start Tivoli Workload Scheduler for z/OS Tracker/Controller/DataStore, enter the dialogs and set up the Controller configuration, and build a workstation, calendar, and application/operation. After completion, you can run a long-term and current plan.

1.11.1 Setting up the initial Controller configuration

Use the following steps to set up the initial Controller configuration:

1. From the primary panel, enter =0.1 to configure the Controller that you will use, as shown in Example 1-13, and press Enter.

Example 1-13 Primary panel testing up the options

```
------------------ OPERATIONS PLANNING AND CONTROL ------------------
 Option ===> =0.1

 Welcome to OPC. You are communicating with TWSC

 Select one of the following options and press ENTER.

 0 OPTIONS          - Define OPC dialog user parameters and options
 1 DATABASE         - Display or update OPC data base information
 2 LTP              - Long Term Plan query and update
 3 DAILY PLANNING   - Produce daily plans, real and trial
 4 WORK STATIONS    - Work station communication
 5 MCP              - Modify the Current Plan
 6 QCP              - Query the status of work in progress
 7 OLD OPERATIONS   - Restart old operations from the DB2 repository

 9 SERVICE FUNC     - Perform OPC service functions
 10 OPTIONAL FUNC   - Optional functions
 X EXIT             - Exit from the OPC dialog
```

2. On the next panel, shown in Example 1-14, you can set up the Controller started task. The entry that is configured is TWSC; other entries are initially from the ISPF table EQQLUDEF, which was configured previously.

Example 1-14 Controller and Server LU name configurations

```
--------------------- OPC CONTROLLERS AND SERVER LU NAMES ---- Row 1 to 4 of 4
Command ===>                                                  Scroll ===> PAGE

Change data in the rows, and/or enter any of the following row commands
I(nn) - Insert, R(nn),RR(nn) - Repeat,  D(nn),DD - Delete

Row    Con-
cmd  S troller Server LU name    Description
'''    _ ____    APPC LUNAME______ TWS FROM REMOTE SYSTEM__
'''    _ OPC0    IS1MEOPV_________ On other_________________
'''    _ OPC0    SEIBM2OO.IS1MEOPV ___________________________
'''    / TWSC    ___________________ TWS on same MVS________
```

3. When this is configured, press PF3 to go back to the Primary Tivoli Workload Scheduler for z/OS panel. (You have completed configuring the Controller options.)

1.12 Building a workstation

To set up the workstation from the primary panel:

1. Enter =1.1.2 on the Option line as shown in Example 1-15, and press Enter.

Example 1-15 Primary option panel

```
------------------ OPERATIONS PLANNING AND CONTROL --------------------
 Option ===> =1.1.2

 Welcome to OPC. You are communicating with TWSC

 Select one of the following options and press ENTER.

 0 OPTIONS           - Define OPC dialog user parameters and options

 1 DATABASE        - Display or update OPC data base information
 2 LTP             - Long Term Plan query and update
 3 DAILY PLANNING   - Produce daily plans, real and trial
 4 WORK STATIONS    - Work station communication
 5 MCP             - Modify the Current Plan
```

```
6 QCP             - Query the status of work in progress
7 OLD OPERATIONS   - Restart old operations from the DB2 repository

9 SERVICE FUNC     - Perform OPC service functions
10 OPTIONAL FUNC    - Optional functions
X EXIT            - Exit from the OPC dialog
```

2. Press Enter on the panel that displays the text in Example 1-16.

Example 1-16 SPECIFYING WORK STATION LIST CRITERIA menu

```
------------------- SPECIFYING WORK STATION LIST CRITERIA --------------------
 Command ===>

 Specify selection criteria below and press ENTER to create a list.

 WORK STATION NAME   ===> ____
 DESTINATION         ===> ________
 TYPE                ===> ___        G , C , P in any combination, or blank
 REPORTING ATTRIBUTE ===> ____       A , S , C , N  in any combination or blank
 FT Work station     ===> _          Y , N or blank
```

3. This opens the Create Workstation panel. Enter `create` on the Command line, as shown in Example 1-17, and press Enter.

Example 1-17 LIST OF WORK STATION DESCRIPTIONS menu

```
---------------------- LIST OF WORK STATION DESCRIPTIONS --- Row 1 to 14 of 57
 Command ===> create                                      SCROLL ===> PAGE

 Enter the CREATE command above to create a work station description or enter
 any of the following row commands:
 B - Browse, D - Delete, M - Modify, C - Copy.

 Row  Work station                              T  R  Last update
 cmd  name  description                              user      date        time
 '    AXDA  dallas.itsc.austin.ibm.com          C  N  TWSRES3   04/06/14    09.08
 '    AXHE  helsinki.itsc.austin.ibm.com        C  N  TWSRES3   04/06/14    09.08
 '    AXHO  Houston.itsc.austin.ibm.com         C  N  TWSRES3   04/06/14    09.09
 '    AXMI  milan.itsc.austin.ibm.com           C  N  TWSRES3   04/06/14    09.09
 '    AXST  stockholm.itsc.austin.ibm.com       C  N  TWSRES3   04/06/14    09.09
 '    CPU   Default Controller Workstation      C  A  TWSRES3   04/06/29    23.38
 '    CPUM  asd                                 C  A  TWSRES4   05/02/11    16.36
 '    CPU1  Default Controller Workstation      C  A  TWSRES4   05/02/10    17.43
```

4. Enter CPU1 for the workstation name, C for workstation type, A for reporting attributes, and the destination name of twscmem, as shown in Example 1-18.

Example 1-18 CREATING GENERAL INFORMATION ABOUT A WORK STATION menu

```
-------------- CREATING GENERAL INFORMATION ABOUT A WORK STATION --------------
Command ===>

Enter the command R for resources  A for availability or M for access method
above, or enter data below:

WORK STATION NAME   ===> cpu1
DESCRIPTION       ===> Initial setup work station____________________
WORK STATION TYPE   ===> c         G  General, C  Computer, P  Printer
REPORTING ATTR      ===> a         A  Automatic, S  Manual start and completion
                                   C  Completion only, N  Non reporting
FT Work station     ===> N         FT Work station, Y or N
PRINTOUT ROUTING    ===> SYSPRINT  The ddname of daily plan printout dataset
SERVER USAGE        ===> N         Parallel server usage C , P , B or N

Options:
 SPLITTABLE         ===> N         Interruption of operation allowed, Y or N
 JOB SETUP          ===> N         Editing of JCL allowed, Y or N
 STARTED TASK, STC  ===> N         Started task support, Y or N
 WTO                ===> N         Automatic WTO, Y or N
 DESTINATION        ===> TWSCMEM_  Name of destination
Defaults:
 TRANSPORT TIME     ===>  0.00     Time from previous work station  HH.MM
```

5. Press PF3 and look for the workstation created in the top-right corner. You have completed building a workstation.

1.12.1 Building a calendar

Use the following steps to build a calendar:

1. Type =1.2.2 on the option line, as shown in Example 1-19, and press Enter.

Example 1-19 Building a calendar

```
------------------- OPERATIONS PLANNING AND CONTROL -------------------
Option ===> =1.2.2

Welcome to OPC. You are communicating with TWSC

Select one of the following options and press ENTER.
```

```
0 OPTIONS         - Define OPC dialog user parameters and options

1 DATABASE        - Display or update OPC data base information
2 LTP             - Long Term Plan query and update
3 DAILY PLANNING   - Produce daily plans, real and trial
4 WORK STATIONS    - Work station communication
5 MCP             - Modify the Current Plan
6 QCP             - Query the status of work in progress
7 OLD OPERATIONS   - Restart old operations from the DB2 repository

9 SERVICE FUNC     - Perform OPC service functions
10 OPTIONAL FUNC   - Optional functions
X EXIT            - Exit from the OPC dialog
```

2. Type `create`, as shown in Example 1-20, and press Enter.

Example 1-20 MODIFYING CALENDARS menu

```
----------------------------- MODIFYING CALENDARS ---------- Row 1 to 13 of 21
Command ===> create                                          Scroll ===> PAGE

Enter the CREATE command above to create a new calendar or
enter any of the following row commands:
B - Browse, C - Copy, D - Delete, M - Modify,
or G to display a calendar graphically

Row Calendar         Description                    Last update
cmd id                                              user     date     time
'   ALLWORKDAYS      default calendar               TWSRES4  05/02/11 16.39
'   BPICAL1          default calendar               TWSRES4  05/02/11 16.39
'   DAILY            Workday Calendar               TWSRES3  04/06/11 16.07
'   DEFAULT          DEFAULT CALENDAR               TWSRES8  04/07/03 13.59
'   DEFAULTMF1       DEFAULT CALENDAR               TWSRES4  05/02/11 16.39
'   HOLIDAYS         Default calendar with holidays TWSRES4  05/02/11 16.39
'   IBM$CALENDAR     default calendar               TWSRES4  05/02/11 16.39
'   INTIALIZE        calendar for initialize        TWSRES1  05/09/16 16.33
```

3. This panel builds the calendar. Type a calendar ID, a description, and a workday end time, as shown in Example 1-21, and press Enter.

Example 1-21 CREATING A CALENDAR menu

```
----------------------------- CREATING A CALENDAR ------------ Row 1 to 1 of 1
 Command ===>                                                 Scroll ===> PAGE

 Enter/change data below and in the rows,
```

```
and/or enter any of the following row commands:
I(nn) - Insert, R(nn),RR(nn) - Repeat, D(nn),DD - Delete

CALENDAR ID        ===> intialize_______
DESCRIPTION        ===> calendar for initialize_______
WORK DAY END TIME  ===> 23.59

Row  Weekday or     Comments                       Status
cmd  date YY/MM/DD
'''' monday________ initialize_________________      w
```

4. Pressing PF3 returns you to the previous panel, where Calendar has been added in the top-right corner.

1.12.2 Building an application/operation

An application contains the tasks that you want Tivoli Workload Scheduler for z/OS to control, such as running a job, issuing a WTO (Write To Operator), or even preparing JCL for a job, which are called operations. Simply, an application is a group of related operations (or jobs). The operations in each application are always run together, so when an operation in the application is run, all operations must also run. An application can also be related operations for a specific task; for example, an application called Daily Planning could have the long-term plan and current plan job run. Other good examples for applications are for Payroll, Accounting, and so forth.

1. Figure 1-11 on page 43 is the starting point for creating an application in Tivoli Workload Scheduler for z/OS (enter option `=1.4.2`).

   Before we go into the RUN and OPER command, we define our application:

   The Application ID can be from one to 16 alphanumeric characters in length, but the first character must be alphabetic.

   The Application TEXT field is optional, but is important in giving a more detailed description of your application. This field can be up to 24 characters.

   The Application TYPE field gives you two options:

   – A for application

   – G for group definition

   We want to keep this an application, so we type `A`.

   Under Owner ID you can insert your own user ID or another ID that is specific to your environment. This field is required (up to 16 characters in length) and it gives you a convenient way to identify applications. The owner ID itself makes a very useful search argument as well; for example, if you had all Payroll applications with an owner ID of PAYNOW, then the owner ID of

PAYNOW can be used as a selection criteria in the Tivoli Workload Scheduler for z/OS panels and reports.

The Owner TEXT is optional and can contain up to 24 characters; it is used to help identify the Owner ID.

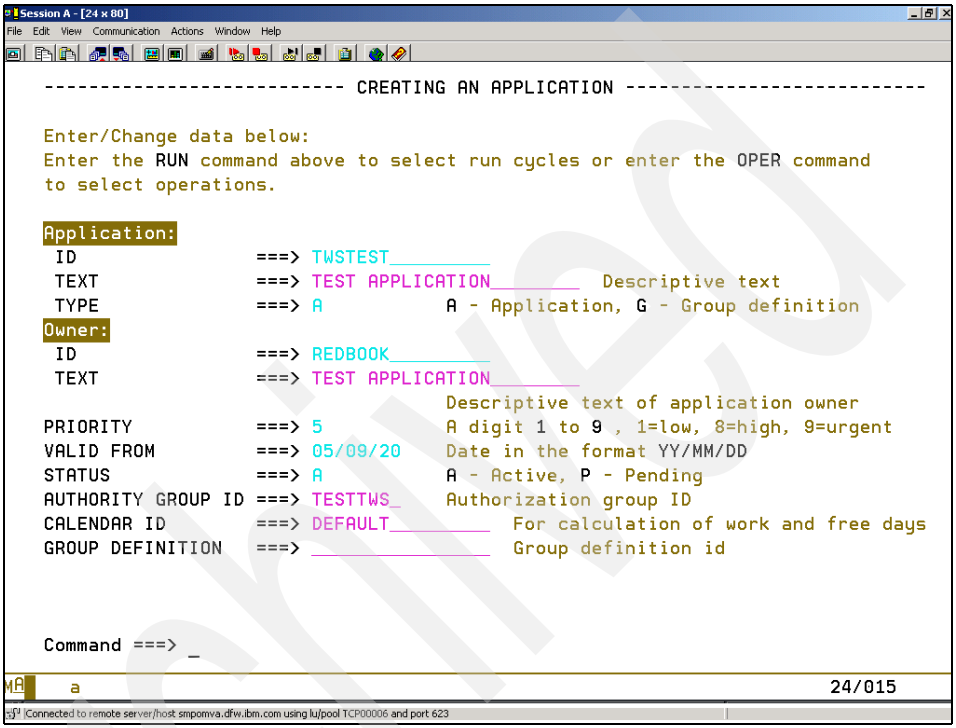


Figure 1-11 Creating an Application - option =1.4.2

The PRIORITY field specifies the priority of the main operation, from 1 being the lowest to 9 being the most urgent. Set PRIORITY to 5.

Important: VALID FROM specifies when the application is available for scheduling. This is a *required* field, and if nothing is entered Tivoli Workload Scheduler for z/OS defaults to the current date.

The status of the application gives you two options:

A Active (can be selected for processing)
P Pending (cannot be selected for processing)

Set STATUS to active (A).

The AUTHORITY GROUP ID is optional, can be from one to eight characters in length, and can be used for security grouping and reporting.

The CALENDAR ID is up to eight characters long and is optional. This field is important, especially if you have several calendars built and the operations have to use a specific calendar. If no calendar is provided in this field, then the DEFAULT calendar is assigned to this application.

The GROUP DEFINITION is used for the calendar and generation of run cycles. The group definition is valid only for applications and is mutually exclusive with the specification of calendar and run cycle information. The group definition field is optional.

Note: As you input information in the required fields, you may press Enter and see a message in the upper-right corner saying `No Operation Found`, and the `A` next to TYPE will be blinking. This is not a cause for alarm; it is just Tivoli Workload Scheduler for z/OS indicating that there are no operations in this application.

2. Type `RUN` on the Command line to build the run cycle.
3. As shown in Figure 1-12 on page 45, type `S` for the row command, designate the application name and application text, and press PF3.

   These are the row command options:

   I Stands for Insert; builds an additional run cycle for you to create.

   R Repeats all of the information in the run cycle you have created (so that you can keep the same run cycle and perform some small edits with very little typing).

   D Deleting the run cycle.

   S For selecting the run cycle and making modifications to the run cycle. (See Figure 1-13 on page 47).

   Name of period/rule is the actual name of our run cycle; this is a good way to be descriptive about the run cycle you are creating. For example, we named our rule `DAILY` because this will run every day. (We clarify that in the Text below it: `Sunday - Saturday`.) The rule can be up to eight characters long with an alphabetic first character. The descriptive text below it can be up to 50 characters in length and is optional.

   Input HH:MM specifies the arrival time of the application. This time determines when occurrences of the application will be included in the current plan, based on the current planning period. This input arrival time is used by Tivoli Workload Scheduler for z/OS when external dependencies are established.

Deadline Day tells when the application should complete. The offset is between 0-99, so 0 indicates that the application should complete on the same day as its start date, 01 would be for the following day, and so on.

The Deadline Time is when you expect the application to end or complete. Depending on how large the application is you may need to give it ample time to complete, but for our example we give it a full 24 hours to complete.

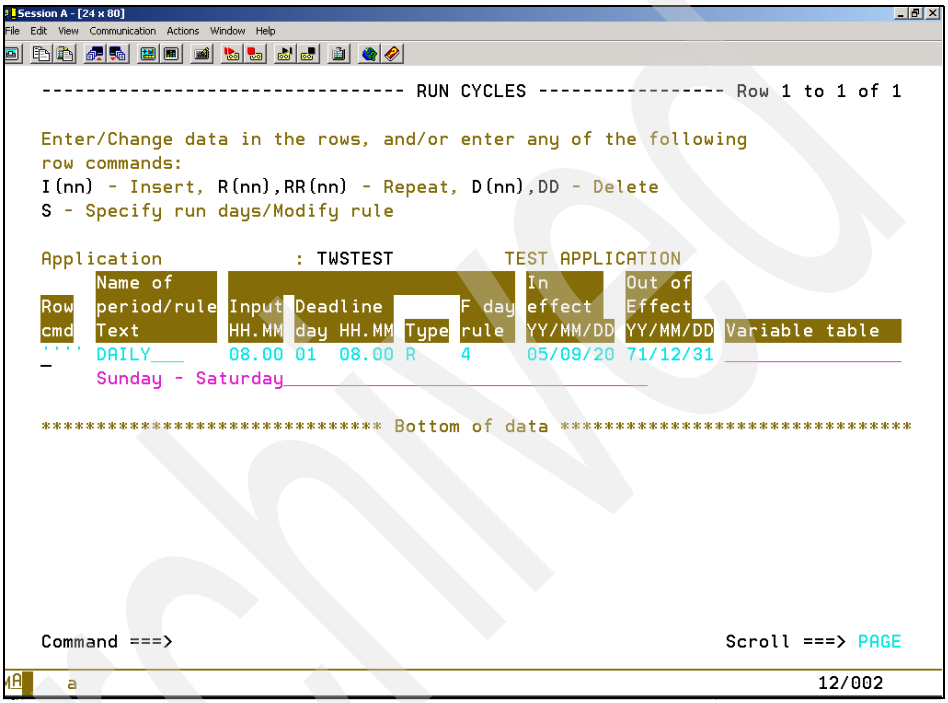


Figure 1-12 Sample run cycle

The Type of run cycle we want for our application is required:

R The Rule-based run cycle, which enables you to select the days that you want the application to run. (We chose R for our example.)

E An Exclusion Rule–based run cycle that enables you to select the days that you do not want the application to run. For example, you can state in the Rule-based run cycle per our example (Figure 1-12) to run daily Monday through Sunday, but then create an Exclusion Rule–based run cycle so the application does not run on a certain day of the month or whatever else you want.

N Offset-based normal run cycle identifies days when the application is to run. This is defined as a cyclic or non-cyclic period defined in the calendar database.

X Offset-based negative run cycle that identifies when the application should not run for the specified period.

The F day rule (freeday run) is used to help identify the days in which the application should run. Selecting E under Type excludes all freedays based on the calendar you chose when building the application initially. So it will only count workdays when you are performing a numeric offset. For example, if you were to define offset 10 in a monthly non-cyclic period, or the 10th day of the month in a Rule-based run cycle, and you select E as the freeday rule, then occurrences are generated on the 10th work day of the month as opposed to the 10th day. If an occurrence is generated on a freeday, it can be handled using these options:

1 Moves to the closest workday *before* the freeday; for example, you may have an application that runs on Thursdays and if there is a Holiday (freeday) for a given Thursday then this application will run on Wednesday, the day before the freeday.

2 Moves to the closest day *after* the freeday, so as described above for option 1, instead of the application running on Thursday it would run the day after the freeday, which would be Friday.

3 Enables the application to run on the freeday. So if you have a calendar defined as Monday through Friday, and you specify in your run cycle to have the application run Monday through Sunday, this option enables you to do that and permits the application to run on any other freedays such as Holidays that may be defined in the calendar.

4 Prevents the application from running on the freeday. In a sense this application points to the calendar and runs according to the calendar setup. So this application will not run on the holidays (freedays) defined in the calendar.

In Effect and Out of Effect specify when the run cycle should start and end. This is good for managing your run cycles and enabling you to have proper maintenance on when it should start and end. More on the Variable Table can be found in Chapter 10, “Tivoli Workload Scheduler for z/OS variables” on page 235.

4. This takes you to the Modifying a Rule panel (Figure 1-13 on page 47), which is the rule we just created called Daily. Three columns identify our rule, and you choose by putting an S in the blank space. For Frequency, we chose **Every**; in the Day column, we selected **Day**, and for the Cycle Specification we chose **Month** because we want this application to run Every Day of the Month.

   With the fields completed, we use the GENDAYS command to see how this works out in the calendar schedule.

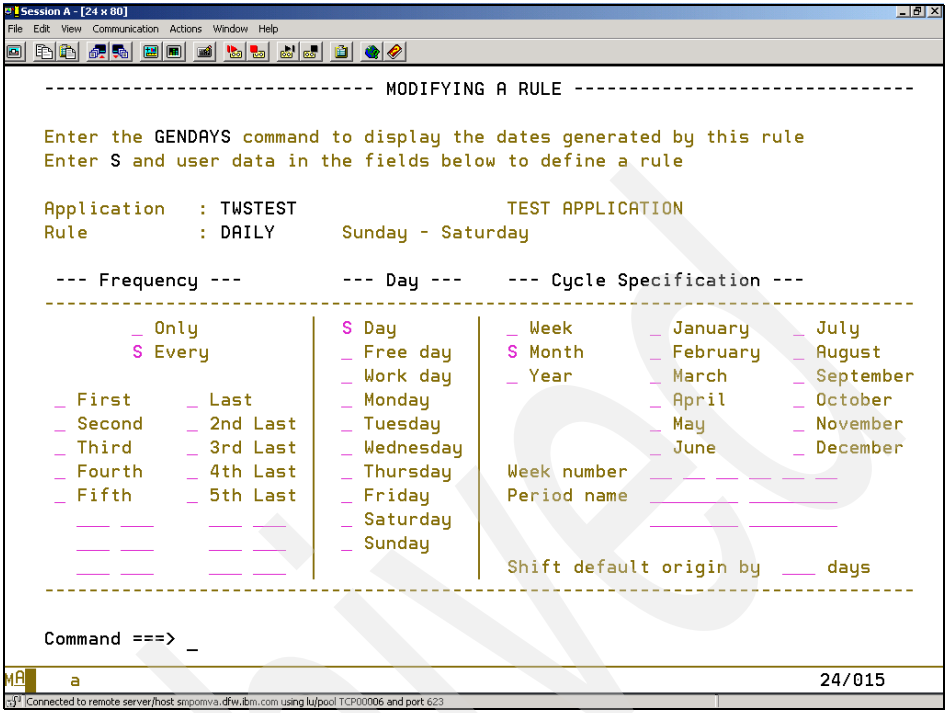


Figure 1-13 Modifying a Rule for an Application

5. Figure 1-14 on page 48 shows the results from the GENDAYS command for the rule that we created in step 4 on page 46 (Figure 1-13). The first thing to notice is the VALID RULE box in the lower-middle part of the panel. This tells us that the rule we created was good. Press Enter and that box goes away. GENDAYS gives us an initial six-month block showing when the job will be scheduled. The start Interval is 05/09/20; recall that was our In Effect Date when we started to create the rule. Everything prior to this date is dark and every date on and after 05/09/20 is lighter; this tells us that these dates have already past. You can press PF8 to scroll forward, PF7 to scroll backward, and pressing PF3 to exit.

   Looking back at Figure 1-13, more can be done with our rule. You can have Every selected for Frequency and choose from the First or Last column. If we add a selection of First but keep everything else the same, our rule changes significantly: Instead of Every Day of the Month, we would have Every First Day of the Month. We can continue to do that for each selection under Frequency: where we choose the First column we go forward from the start of the month and when we choose Last we go backward starting from the end of the month. It goes up to the Fifth in the First column and 5th Last in the Last Column, and you can add more with the blanks and using numerics only, so

for sixth day we would enter 006 in one of the blanks under First, likewise for Last. You can also use Only, for example, to force the rule to run on Only the First Day of the Month. Many more combinations can be created using Tivoli Workload Scheduler. For more samples refer to *IBM Tivoli Workload Scheduler for z/OS Managing the Workload Version 8.2,* SC32-1263, or just try different rules and use the GENDAYS command to confirm that the rule created is just what you wanted.

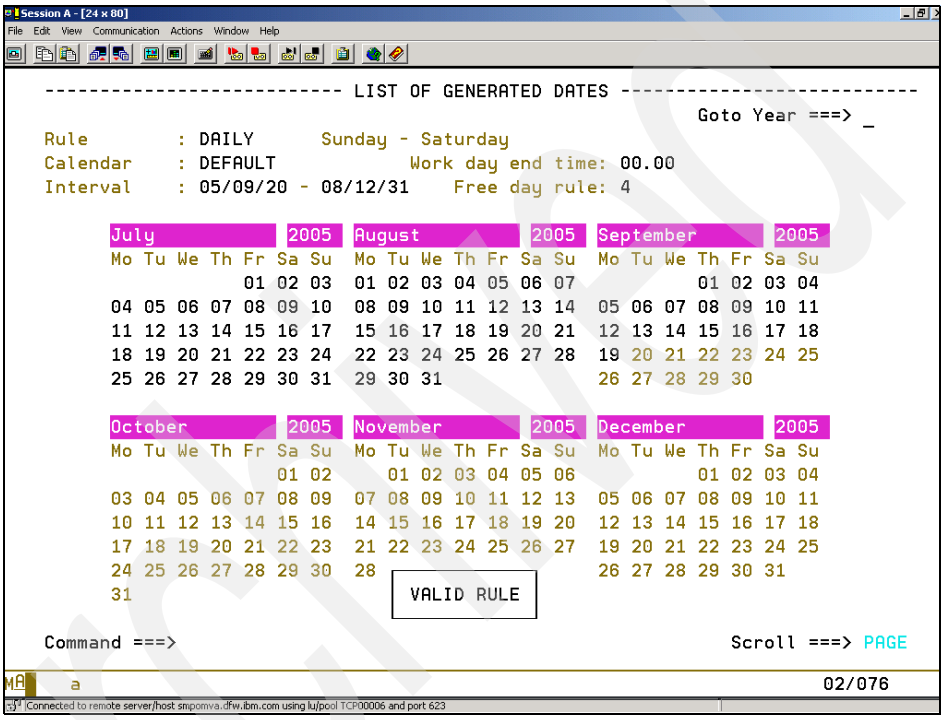


Figure 1-14 GENDAYS results

6. Press PF3 to back out until you reach the Modifying the Application panel, and type OPER on the command line (or just OP depending on how Tivoli Workload Scheduler for z/OS is installed) to insert operations/jobs using the OPERATIONS panel (Figure 1-15 on page 49).

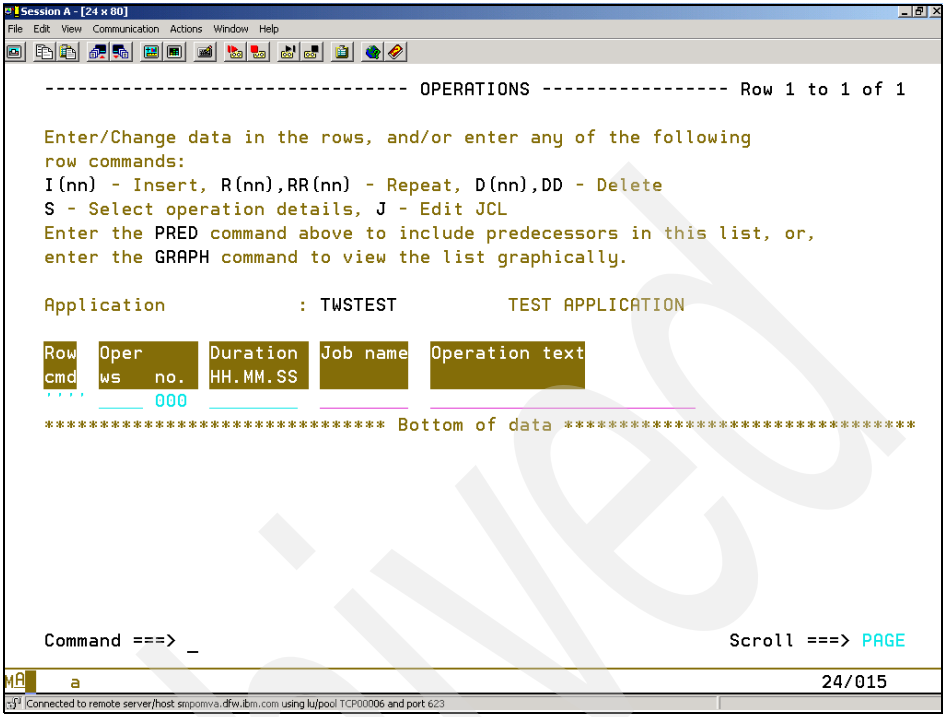


Figure 1-15 OPER command

7. On the OPERATIONS panel:
   - The PRED command lists any operations that are predecessors to this job.
   - Under Oper ws, we insert our workstation name (`CPU1` for our workstation) and operation number (operation number range is 1 - 255; ours is `001`).
   - For Duration we can put in anything we want because eventually Tivoli Workload Scheduler for z/OS will adjust this figure dynamically from its experience of the actual durations. We enter `00.01.00`.
   - The job name is, of course, the name of the job you want to run. (We use `TWSTEST`). Be sure that the job is in a library that is part of the Tivoli Workload Scheduler for z/OS concatenation.
   - Operation Text is a short description of the job itself (such as `testing`).

   Everything is set up for our job, we have defined the run cycle and set up the operation, so now we ensure that it submits and only runs at the time we specified. For the row command, we enter `S.4`.
8. This opens the JOB, WTO, AND PRINT OPTIONS panel (Figure 1-16 on page 50), which shows our application name, operation name, and job name.

The important part is under Job Release Options, where we want Tivoli Workload Scheduler for z/OS to be able to submit this job when it comes up in the current plan, so we set SUBMIT to Y. We want the job to run when we want it to and not before, so we set the TIME DEPENDENT to Y. This way, when the long-term plan and the current plan run for the day, the application does not start until 8:00 a.m. as we specified in the Run Cycle.

When you have entered the parameters, press PF3.

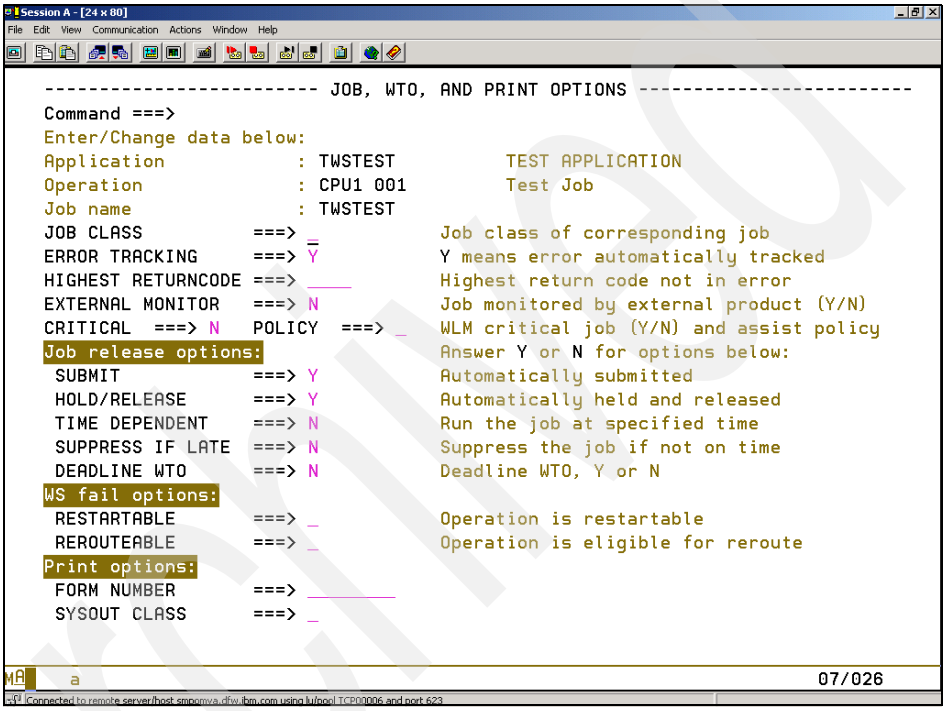


Figure 1-16 JOB, WTO, AND PRINT OPTIONS menu

The workstation has been created, a calendar was created, and an application/operation is now in the database. The next step is to create the current plan, but first we need to create a long-term plan.

1.12.3 Creating a long-term plan

In this section, we create, modify, and extend the Tivoli Workload Scheduler for z/OS long-term plan from the Tivoli Workload Scheduler for z/OS panels.

1. Enter =2.2 on the command line.

   From anywhere in Tivoli Workload Scheduler, enter the command option =2, and then select option 2 (BATCH) to modify, create and extend the long-term plan (Figure 1-17).

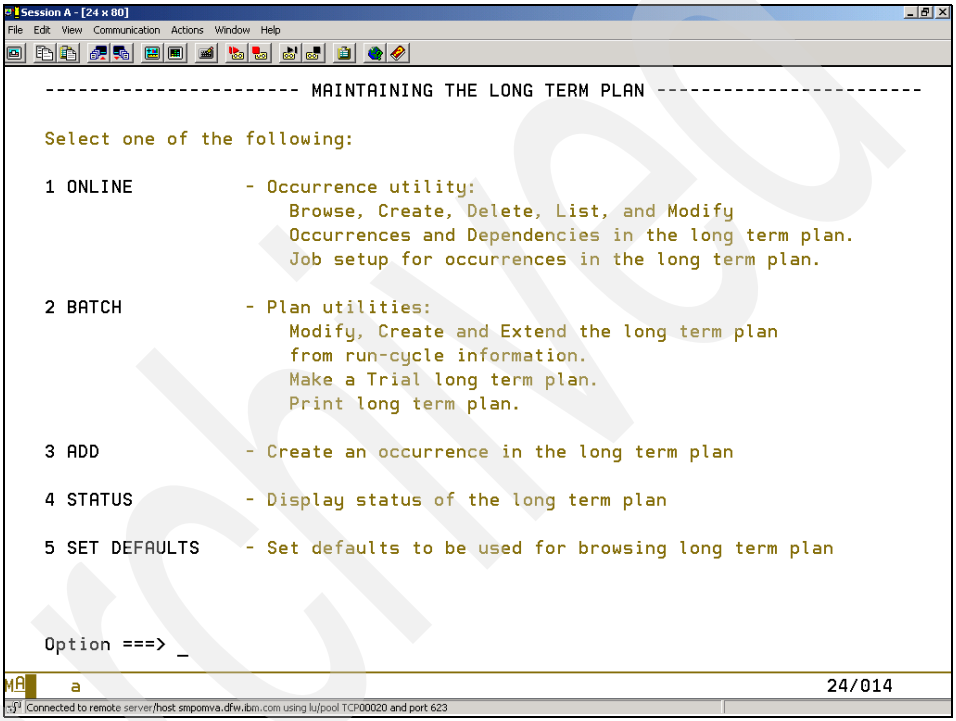


Figure 1-17 Maintaining the long-term plan (panel =2.2)

2. Because this will be our first time, we must create a long-term plan. From the next menu, choose option 7 and press Enter to create a new long-term plan.

3. Enter the start and end dates of the long-term plan (Figure 1-18 on page 52).

   Our START is the current day, and END can be any date of your choosing. Here we chose a five-week end date, but you can make it any date you desire up to four years. It is good practice to choose a date of five weeks and then you can always modify it further in the future. Enter the end date (current day + 5 weeks) and press Enter.

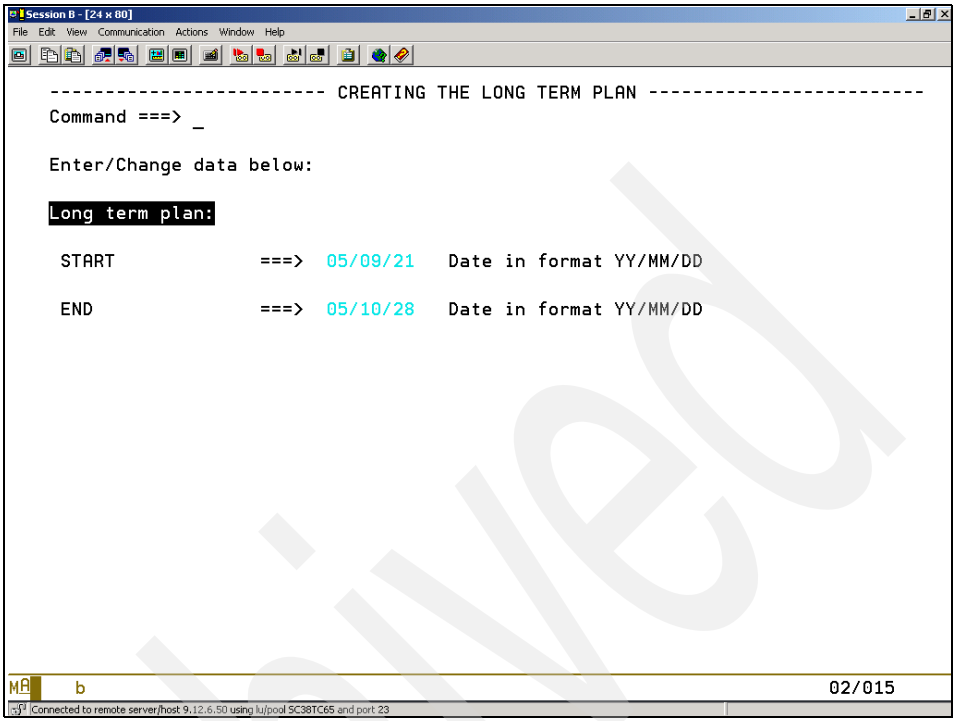


Figure 1-18 Creating the long-term plan

4. Generate the batch job for the long-term plan (Figure 1-19 on page 53). Insert a valid job card, enter `E` for Edit, and press Enter. This enables you to check the skeletons and make sure they are set up properly.

   You can also edit the data set name or allow the default name chosen by Tivoli Workload Scheduler. When you are in Edit you can submit the job by typing `submit` on the command line. Or if you are sure the skeletons are properly set up, choose `S` (for submit) under the Submit/Edit field and press Enter. You will see the message `JOB TWSRES1A(JOB04831) SUBMITTED`.

5. When the batch job finishes, check the JCL for errors. The return code should be an RC of 0 on the completed job. If the long-term plan is created, you can scan the scheduled occurrences most easily online using option 1 (ONLINE) from the LTP menu. If things are not what you expect, you can change occurrences using this panel, but it is easier, while you have no current plan, to correct the database and re-create the long-term plan. You cannot re-create the long-term plan when you have a current plan; you have to delete the current plan first with the REFRESH function.

If the long-term plan looks good, put the jobs that you need into the EQQJBLIB data set. The member name must be the same as the operation name.

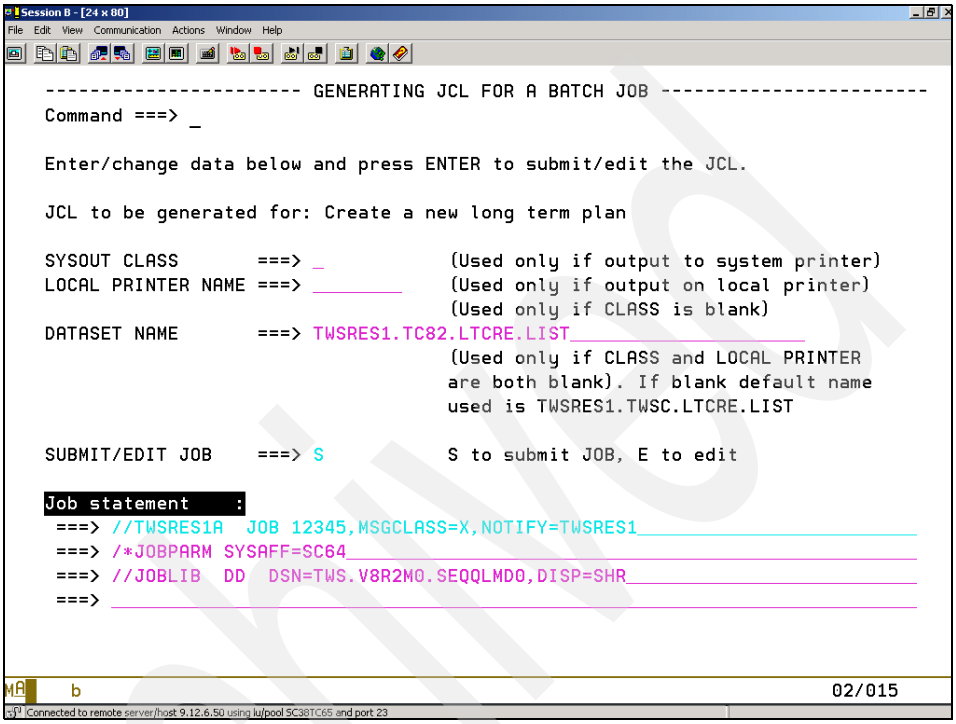


Figure 1-19 Generating JCL for a batch job

1.12.4 Creating a current plan

With our long-term plan created, we can now create the current plan by following these steps:

1. Enter `=3.2` on the command line and press Enter.

   Select option `3` (Daily Planning) from the main menu. Now from the Producing OPC Daily Plans menu select option `2` (Extend).

2. This opens the Extending Current Plan Period panel (Figure 1-20). Type in the values as shown (except for the date and time, which can be anything).

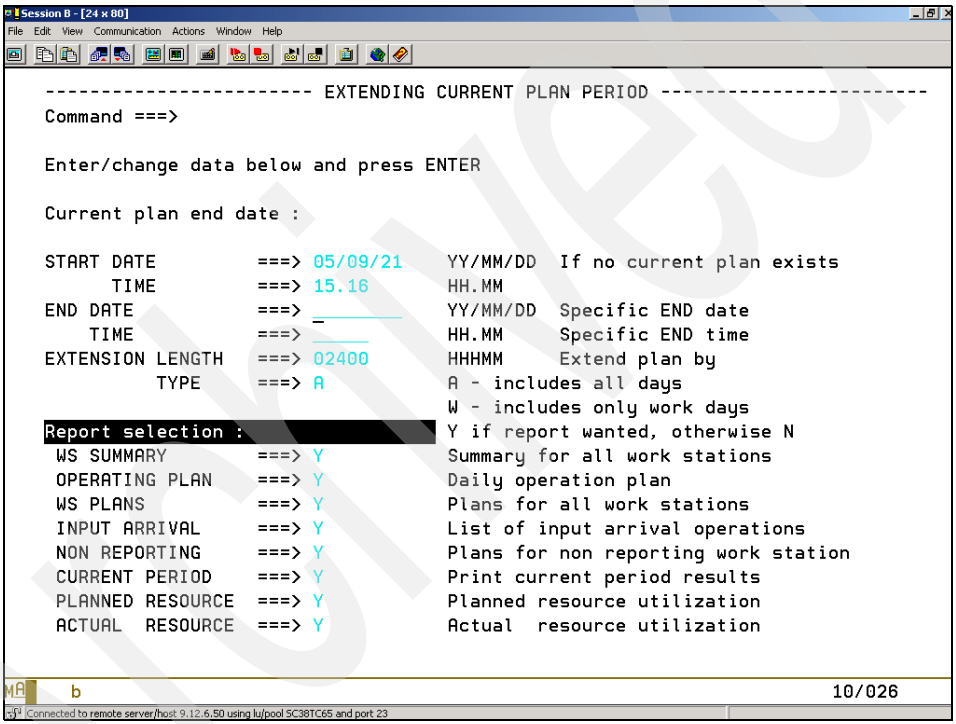


Figure 1-20 Extending current plan period

Enter the batch parameters as on the long-term plan to submit the job and check the output for error messages. The Return code should be RC 4 or less.

The current plan can span from 1 minute to 21 days from the time of its creation or extension. Tivoli Workload Scheduler for z/OS brings in from the long-term plan all occurrences with an input arrival time that are within the period you specify. Tivoli Workload Scheduler for z/OS then creates a detailed schedule for the operations that are contained in these occurrences.

If you are extending the current plan, Tivoli Workload Scheduler for z/OS also carries forward into the new plan any uncompleted occurrences in the existing plan. Therefore, the current plan could contain information about occurrences back to the creation of the plan if there are uncompleted occurrences. To determine what time span your current plan should have, consider:

– The longer the plan, the more computing resources required to produce it.
– Changes in the long-term plan are reflected in the current plan only after the current plan is extended.
– You cannot amend occurrences in the long-term plan that have an input arrival time before the end of the current plan.
– Plans longer than 24 hours will contain two occurrences of daily applications and can cause confusion for the operations staff.

> **Note:** The maximum duration of the current plan is 21 days.

– Current plan must be extended more frequently.
– Current plan can contain a maximum of 32,760 application occurrences.

Now, you have completed the installation. You can proceed to Chapter 2, “Tivoli Workload Scheduler for z/OS installation verification” on page 57 for the verification of your installation.

2

Tivoli Workload Scheduler for z/OS installation verification

After completing the installation, you must verify the functions of each of the started tasks. This chapter discusses in detail how to do this, and gives troubleshooting hints to resolve issues. You must verify that all the started tasks have started correctly, that the functions of the started tasks are working correctly, and that they are communicating between each other. This requires you to analyze the MLOGs, submit a job, and verify that the event data set, DataStore, and restart are working correctly.

This chapter covers the following:

- Verifying the Tracker
- Controller checkout
- DataStore checkout

2.1 Verifying the Tracker

Now is the time to double-check the installation performed in Chapter 1, “Tivoli Workload Scheduler for z/OS installation” on page 3. Make sure that every step is complete, the data sets have been allocated, and the communication mechanism is in place.

After starting the Tracker (and when troubleshooting), the MLOG is the first place to verify that the Tracker is working correctly.

Important: The MLOG is a valuable tool for troubleshooting and should be inspected with great detail when trouble is experienced. If the started task does not initiate and stay running, check the MLOG and syslog for errors.

2.1.1 Verifying the MLOG

After the Tracker has started, verify the parameters at the beginning of the MLOG. Confirm that all initialization parms got an RC 0 when the Tracker was started by searching for the EQQZ016I MESSAGE. If there are errors, correct and restart the Tracker. The initialization statement might look like Example 2-1.

Example 2-1 One Initialization parameter

```
EQQZ013I NOW PROCESSING PARAMETER LIBRARY MEMBER TWST
EQQZ015I INIT STATEMENT: OPCOPTS  OPCHOST(NO)
EQQZ015I INIT STATEMENT:         ERDRTASK(0)
EQQZ015I INIT STATEMENT:       EWTRTASK(YES)    EWTRPARM(STDEWTR)
EQQZ015I INIT STATEMENT:         JCCTASK(NO)
EQQZ015I INIT STATEMENT: /*     SSCMNAME(EQQSSCMF,TEMPORARY)  */
EQQZ015I INIT STATEMENT: /*      BUILDSSX(REBUILD) */
EQQZ015I INIT STATEMENT: EQQZ015I INIT STATEMENT:          ARM(YES)
EQQZ016I RETURN CODE FOR THIS STATEMENT IS: 0000
```

Verify that all started subtasks that are needed are configured. Ensure that all required subtasks are active by looking for EQQZ005I (subtask starting) and EQQSU01I (subtask started).

The data router and submit tasks are always started. You should see the messages in Example 2-2 on page 59.

Example 2-2 Subtask starting

```
EQQZ005I OPC SUBTASK DATA ROUTER TASK IS BEING STARTED
EQQF001I DATA ROUTER TASK INITIALIZATION IS COMPLETE
EQQZ005I OPC SUBTASK JOB SUBMIT TASK IS BEING STARTED
EQQSU01I THE SUBMIT TASK HAS STARTED
```

Also, verify that the Tracker has started an Event Writer. You should see these messages:

```
EQQZ005I OPC SUBTASK EVENT WRITER IS BEING STARTED
EQQW065I EVENT WRITER STARTED
```

Examine MLOG for any error messages.

Important: The first time the Event Writer is started and you are examining error messages in the MLOG, you will see an error message with a SD37 when the Event data set is formatted. This is normal and should be ignored.

If you see error messages in the message log for an Event Reader or an NCF connection, this is because you cannot fully verify an Event Reader function or NCF connection until the Controller is active and a current plan exists. Active Tracker-connection messages for XCF connections are written to the Controller message log when the Controller is started.

Examine the MLOG for being complete. If it seems incomplete the output may still be in a buffer. If you are unsure whether the log is complete, issue a dummy modify command like this:

```
F ssname,xx
```

Message EQQZ049E is written to the log when the command is processed. This message will be the last entry in the log.

2.1.2 Verifying the events in the event data set

The event data set contains all the events that the Tracker is recording. This is a wrap data set; when end of file is reached it will start writing from the beginning of the data set.

The event data set is needed to even out any difference in the rate that events are being generated and processed and to prevent events from being lost if the Tivoli Workload Scheduler for z/OS address space or a Tivoli Workload Scheduler for z/OS subtask must be restarted. The first byte in an exit record is A if the event is created on a JES2 system, or B if the event is created on a JES3 system. This byte is found in position 21 of a standard event record, or position

47 of a continuation (type N) event. Bytes 2 and 3 in the exit record define the event type. These event types are generated by Tivoli Workload Scheduler for z/OS for jobs and started tasks (Example 2-3).

Example 2-3 Tracker events

```
1 Reader event. A job has entered the JES system.
2 Job-start event. A job has started to execute.
3S Step-end event. A job step has finished executing.
3J Job-end event. A job has finished executing.
3P Job-termination event. A job has been added to the JES output queues.
4 Print event. An output group has been printed.
5 Purge event. All output for a job has been purged from the JES system.
```

If any of these event types are not being created in the event data set (EQQEVDS), a problem must be corrected before Tivoli Workload Scheduler for z/OS is started in production mode.

Notes:

- The creation of step-end events (3S) depends on the value you specify in the STEPEVENTS keyword of the EWTROPTS statement. The default is to create a step-end event only for abending steps in a job or started task.
- The creation of print events depends on the value you specify in the PRINTEVENTS keyword of the EWTROPTS statement. By default, print events are created.

To test whether you are getting all of your events, you can submit a simple job (Example 2-4) from Tivoli Workload Scheduler. After the print purges, you can see whether you have all the events.

Note: If you do not print the output, you will not get an A4 event.

Example 2-4 Test job

```
//JOBA JOB .............
//VERIFY EXEC PGM=IEBGENER
//*
//SYSPRINT DD DUMMY
//SYSUT2 DD SYSOUT=A
//SYSIN DD DUMMY
//SYSUT1 DD *
SAMPLE TEST OUTPUT STATEMENT 1
//*
```

In Example 2-5 you can see what the EV data set looks like. Note to get this view you can perform an **X all** and a **find all** on the job name or job number. It is a good way to see whether the Tracker/exits are producing all of the events. Note this is a JES2 system because they are showing as Ax events. If it were JES3, it would be Bx events.

Example 2-5 Tivoli Workload Scheduler for z/OS events in EV data set

```
VIEW       TWS.INST.EV64                                      Columns 00001 00072
Command ===>                                                    Scroll ===> CSR
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
****** ***************************** Top of Data ******************************
=====>    t       ä        A1   -  SMPAPPLYJOB04865   | *'f    | *'=
=====>    u       ä        A2      SMPAPPLYJOB04865   | *'ý    | *'=    | *'ý
=====>    v       ä        A3J  -  SMPAPPLYJOB04865   | *eÙ    | *'=    | *'ý   |
=====>    w       ä        A3P Ø-  SMPAPPLYJOB04865   | *f     | *'=    | *'«   |
=====>    ^       ä        A5      SMPAPPLYJOB04865   | )      | *'=    | *f    |
****** ***************************** Bottom of Data ***************************
```

An indication of missing events can also be seen from the Tivoli Workload Scheduler for z/OS ISPF panels showing a status of S (showing submitted and then it shows no further status even though the job has completed).

2.1.3 Diagnosing missing events

Problem determination depends on which event is missing and whether the events are created on a JES2 or JES3 system. In Table 2-1, the first column is the event type that is missing, and the second column tells you what action to perform. Events created on a JES2 system are prefixed with A, and events created on a JES3 system with B. The first entry in the table applies when all event types are missing (when the event data set does not contain any tracking events).

Table 2-1 Missing event diagnosis

Event	Problem determination action
All	1. Verify in the EQQMLOG dataset that the event writer has started successfully. 2. Verify that the definition of the EQQEVDS ddname in the IBM Tivoli Workload Scheduler for z/OS started-task procedure is correct (that is, events are written to the correct dataset). 3. Verify that the required exits have been installed. 4. Verify that the IEFSSNnn member of SYS1.PARMLIB has been updated correctly, and that an IPL of the z/OS system has been performed since the update.

Event	Problem determination action
A1	If both A3P and A5 events are also missing: 1. Verify that the Tivoli Workload Scheduler for z/OS version of the JES2 exit 7 routine has been correctly installed. Use the $T EXIT(7) JES command. 2. Verify that the JES2 initialization dataset contains a LOAD statement and an EXIT7 statement for the Tivoli Workload Scheduler for z/OS version of JES2 exit 7 (OPCAXIT7). 3. Verify that the exit has been added to a load module library reachable by JES2 and that JES2 has been restarted since this was done. If either A3P or A5 events are present in the event dataset, call an IBM service representative for programming assistance.
B1	1. Verify that the Tivoli Workload Scheduler for z/OS version of the JES3 exit IATUX29 routine has been installed correctly. 2. Verify that the exit has been added to a load-module library that JES3 can access. 3. Verify that JES3 has been restarted.
A2/B2	1. Verify that the job for which no type 2 event was created has started to execute. A type 2 event will not be created for a job that is flushed from the system because of JCL errors. 2. Verify that the IEFUJI exit has been installed correctly: a. Verify that the SMF parameter member SMFPRMnn in the SYS1.PARMLIB dataset specifies that the IEFUJI exit should be called. b. Verify that the IEFUJI exit has not been disabled by an operator command. c. Verify that the correct version of IEFUJI is active. If SYS1.PARMLIB defines LPALIB as a concatenation of several libraries, z/OS uses the first IEFUJI module found. d. Verify that the library containing this module was updated by the Tivoli Workload Scheduler for z/OS version of IEFUJI and that z/OS has been IPLed since the change was made.
A3S/B3S	If type 3J events are also missing: 1. Verify that the IEFACTRT exit has been installed correctly. 2. Verify that the SMF parameter member SMFPRMnn in the SYS1.PARMLIB dataset specifies that the IEFACTRT exit should be called. 3. Verify that the IEFACTRT exit has not been disabled by an operator command. 4. Verify that the correct version of IEFACTRT is active. If SYS1.PARMLIB defines LPALIB as a concatenation of several libraries, z/OS uses the first IEFACTRT module found. 5. Verify that this library was updated by the IBM Tivoli Workload Scheduler for z/OS version of IEFACTRT and that z/OS has been IPLed since the change was made. If type 3J events are not missing, verify, in the EQQMLOG dataset, that the Event Writer has been requested to generate step-end events. Step-end events are created only if the EWTROPTS statement specifies STEPEVENTS(ALL) or STEPEVENTS(NZERO) or if the job step abended.
A3J/B3J	If type A3S events are also missing, follow the procedures described for type A3S events. If type A3S events are not missing, call an IBM service representative for programming assistance.

Event	Problem determination action
A3P	If A1 events are also missing, follow the procedures described for A1 events. If A1 events are not missing, call an IBM service representative for programming assistance.
B3P	1. Verify that the Tivoli Workload Scheduler for z/OS version of the JES3 exit IATUX19 routine has been correctly installed. 2. Verify that the exit has been added to a load-module library that JES3 can access. 3. Verify that JES3 has been restarted.
A4/B4	1. If you have specified PRINTEVENTS(NO) on the EWTROPTS initialization statement, no type 4 events are created. 2. Verify that JES has printed the job for which no type 4 event was created. Type 4 events will not be created for a job that creates only held SYSOUT datasets. 3. Verify that the IEFU83 exit has been installed correctly: a. Verify that the SMF parameter member SMFPRMnn in the SYS1.PARMLIB dataset specifies that the IEFU83 exit should be called. b. Verify that the IEFU83 exit has not been disabled by an operator command. c. Verify that the correct version of IEFU83 is active. If SYS1.PARMLIB defines LPALIB as a concatenation of several libraries, z/OS uses the first IEFU83 module found. d. Verify that the library containing this module was updated by the Tivoli Workload Scheduler for z/OS version of IEFU83 and that z/OS has been IPLed since the change was made. e. For JES2 users (A4 event), ensure that you have not specified TYPE6=NO on the JOBCLASS and STCCLASS statements of the JES2 initialization parameters.
A5	1. Verify that JES2 has purged the job for which no A5 event was created. 2. Ensure that you have not specified TYPE26=NO on the JOBCLASS and STCCLASS statements of the JES2 initialization parameters. 3. If A1 events are also missing, follow the procedures described for A1 events. 4. If A1 events are not missing, call an IBM service representative for programming assistance.
B5	1. Verify that JES3 has purged the job for which no B5 event was created. 2. If B4 events are also missing, follow the procedures described for B4 events. 3. If B4 events are not missing, call an IBM service representative for programming assistance.

2.2 Controller checkout

When verifying the Controller started task, check the installation chapter to verify that all steps have been completed. As in the Tracker, verify in the MLOG that the initialization parameters are getting Return codes of 0 (EQQZ016I).

2.2.1 Reviewing the MLOG

After verifying that there are no initialization parameter errors, you should check the MLOG for the proper subtasks starting. The Controller subtasks will be different from the Tracker subtasks.

Check that all required subtasks are active. Look for these messages when the Controller is started.

Active general-service messages:

- `EQQZ005I OPC SUBTASK GENERAL SERVICE IS BEING STARTED`
- `EQQZ085I OPC SUBTASK GS EXECUTOR 01 IS BEING STARTED`
- `EQQG001I SUBTASK GS EXECUTOR 01 HAS STARTED`
- `EQQG001I SUBTASK GENERAL SERVICE HAS STARTED`

Note: The preceding messages, EQQZ085I and EQQG001I, are repeated for each general service executor that is started. The number of executors started depends on the value you specified on the GSTASK keyword of the OPCOPTS initialization statement. The default is to start all five executors.

Active data-router-task messages:

- `EQQZ005I OPC SUBTASK DATA ROUTER TASK IS BEING STARTED`
- `EQQF001I DATA ROUTER TASK INITIALIZATION IS COMPLETE`

Note: If you do not yet have a current plan, you will receive an error message:

```
EQQN105W NO VALID CURRENT PLAN EXISTS. CURRENT PLAN VSAM I/O IS
NOT POSSIBLE.
```

When you start a Controller and no current plan exists, you will still see a number of EQQZ005I messages each indicating that a subtask is being started. But these subtasks will not start until a current plan is created.

If you have specified an Event Reader function or NCF connections, these tasks will end if no current plan exists.

As in the Tracker you should make sure that you have all of the MLOG by issuing the dummy modify command to the Controller subtask:

```
F TWSC,xx
```

2.2.2 Controller ISPF checkout

During checkout it is easiest if you first verify Tivoli Workload Scheduler for z/OS without RACF being involved. When the Tivoli Workload Scheduler for z/OS

checkout is complete you can begin testing the RACF portion of Tivoli Workload Scheduler. If you have followed 1.11, “Configuring Tivoli Workload Scheduler for z/OS; building a current plan” on page 37, you have already checked out ISPF and determined that you are able to use the ISPF panels in Tivoli Workload Scheduler. If not, you should log on to Tivoli Workload Scheduler for z/OS and, from the primary panel, run `=0.1` and initialize the options as in Example 1-13 on page 37.

From there, explore the other options and make sure you can get to the other panels successfully. If you can, then ISPF is working correctly. If you have set up the RACF profiles and are seeing `user not authorized` in the top-right corner of the panel, review your RACF profiles and make sure that the profiles are set up correctly as demonstrated in Chapter 7, “Tivoli Workload Scheduler for z/OS security” on page 163. When you get a message in the top-right corner, pressing PF1 gives you more detailed information about the error you are seeing.

Verify that the RACF profiles you set up are working correctly and restricting or allowing access as you have prescribed in the profiles.

2.3 DataStore checkout

DataStore is the collector of sysouts that are used for restarts and for browsing syslog from the ISPF panels. Analyze the MLOG as you did for the Tracker and Controller and look for parameter initialization errors. Look for the EQQZ016I with a Return code of 0. If you find errors, correct them and restart the DataStore.

After the Controller has been started, ensure that the messages shown in Example 2-6 appear in the message log. (This example shows messages for an SNA connection.)

Example 2-6 DataStore MLOG messages

```
02/07 12.11.39 EQQZ015I INIT STATEMENT: RCLOPTS CLNJOBPX(EQQCL)
02/07 12.11.39 EQQZ015I INIT STATEMENT: DSTDEST(TWSFDEST)
02/07 12.11.43 EQQPS01I PRE SUBMITTER TASK INITIALIZATION COMPLETE
02/07 12.11.46 EQQFSF1I DATA FILE EQQSDF01 INITIALIZATION COMPLETED
02/07 12.11.46 EQQFSF1I DATA FILE EQQSDF02 INITIALIZATION COMPLETED
02/07 12.11.46 EQQFSF1I DATA FILE EQQSDF03 INITIALIZATION COMPLETED
02/07 12.11.46 EQQFSI1I SECONDARY KEY FILE INITIALIZATION COMPLETED
02/07 12.11.46 EQQFSD5I SYSOUT DATABASE INITIALIZATION COMPLETE
02/07 12.11.46 EQQFL01I JOBLOG FETCH TASK INITIALIZATION COMPLETE
02/07 12.11.46 EQQFSD1I SYSOUT DATABASE ERROR HANDLER TASK STARTED
02/07 12.11.46 EQQFV36I SESSION I9PC33A3-I9PC33Z3 ESTABLISHED
```

There should be an EQQFSF1I message for each EQQSDFxx file specified in the startup procedure. There should be an EQQFV36I message for each SNA connection. Verify that the DSTDEST for message EQQZ015I matches the SYSDEST in the DataStore message log.

For XCF, you will see this message:

```
09/27 11.14.01 EQQFCC9I XCF TWSDSC64 HAS JOINED XCF GROUP TWS82GRP
```

The primary function of DataStore is to retrieve sysout from the JES SPOOL and save it in the DataStore repository. To test this function, submit a job (in this case NEOJOB in application NEOAP). Issue a =5.3 from the command line from the Tivoli Workload Scheduler for z/OS Primary Menu to display Example 2-7.

Example 2-7 Selecting operations

```
--------------------------- SELECTING OPERATIONS -----------------------------
Command ===>

Specify selection criteria below and press ENTER to create an operation list.

JOBNAME            ===> NE*_____
FAST PATH          ===> N               Valid only along with jobname
                                        Y Yes, N No
APPLICATION ID     ===> ________________
OWNER ID           ===> ________________
AUTHORITY GROUP    ===> ________
WORK STATION NAME  ===> ____
PRIORITY           ===> _               Low priority limit
MANUALLY HELD      ===> _               Y Yes, N No
STATUS             ===> _________       Status codes list:
                                         A R * S I C E W U and D
Input arrival in format YY/MM/DD  HH.MM
 FROM              ===> ________  _____
 TO                ===> ________  _____
GROUP DEFINITION   ===> ________________
CLEAN UP TYPE      ===> ____            Types  list: A M I N or blank
CLEAN UP RESULT    ===> __              Results list: C E or blank
OP. EXTENDED NAME  ===> ________________________________________________
```

Press Enter to display Example 2-8.

Example 2-8 Modifying operations in the current plan

```
----------------- MODIFYING OPERATIONS IN THE CURRENT PLAN -- Row 1 to 1 of 1
 Command ===>                                               Scroll ===> PAGE

 Enter the GRAPH command above to view list graphically,
 enter the HIST command to select operation history list, or
```

```
 enter any of the following row commands:
 J   - Edit JCL                     M   - Modify       B   - Browse details
 DEL - Delete Occurrence            MH  - Man. HOLD    MR  - Man. RELEASE oper
 O   - Browse operator instructions NP  - NOP oper     UN  - UN-NOP oper
 EX  - EXECUTE operation            D   - Delete Oper RG  - Remove from group
 L   - Browse joblog                RC  - Restart and CleanUp
 FSR - Fast path SR                 FJR - Fast path JR
 RI  - Recovery Info

 Row  Application id   Operat   Jobname  Input Arrival  Duration Op Depen   S Op
 cmd                   ws   no.          Date     Time  HH.MM.SS ST Su  Pr    HN
'''
'''' NEOAP            CPU1 010 NEOJOB   05/10/08 00.01 00.00.01 YN   0   0 C NN
 ******************************** Bottom of data
```

If you enter an L in the first column of the row that displays the application, you should see Example 2-9.

Example 2-9 List command

```
------------------ MODIFYING OPERATIONS IN THE CURRENT PLAN -- Row 1 to 1 of 1
 Command ===>                                                Scroll ===> PAGE

 Enter the GRAPH command above to view list graphically,
 enter the HIST command to select operation history list, or
 enter any of the following row commands:
 J   - Edit JCL                     M   - Modify       B   - Browse details
 DEL - Delete Occurrence            MH  - Man. HOLD    MR  - Man. RELEASE oper
 O   - Browse operator instructions NP  - NOP oper     UN  - UN-NOP oper
 EX  - EXECUTE operation            D   - Delete Oper  RG  - Remove from group
 L   - Browse joblog                RC  - Restart and CleanUp
 FSR - Fast path SR                 FJR - Fast path JR
 RI  - Recovery Info

 Row  Application id   Operat   Jobname  Input Arrival  Duration Op Depen   S Op
 cmd                   ws   no.          Date     Time  HH.MM.SS ST Su  Pr    HN
 L''' NEOAP            CPU1 010 NEOJOB   05/10/08 00.01 00.00.01 YN   0   0 C NN
 ******************************** Bottom of data ******************************
```

Press Enter to display the next panel, which shows that the sysout is being retrieved (Example 2-10). Note the message JOBLOG Requested in the top-right corner: The Controller has asked the DataStore to retrieve the sysout.

Example 2-10 Sysout

```
------------------ MODIFYING OPERATIONS IN THE CURREN         JOBLOG REQUESTED
 Command ===>                                                Scroll ===> PAGE
```

```
Enter the GRAPH command above to view list graphically,
enter the HIST command to select operation history list, or
enter any of the following row commands:
J   - Edit JCL                      M   - Modify       B   - Browse details
DEL - Delete Occurrence             MH  - Man. HOLD    MR  - Man. RELEASE oper
O   - Browse operator instructions  NP  - NOP oper     UN  - UN-NOP oper
EX  - EXECUTE operation             D   - Delete Oper  RG  - Remove from group
L   - Browse joblog                 RC  - Restart and CleanUp
FSR - Fast path SR                  FJR - Fast path JR
RI  - Recovery Info

Row  Application id   Operat   Jobname  Input Arrival  Duration Op Depen   S Op
cmd                   ws   no.          Date     Time  HH.MM.SS ST Su  Pr    HN
'''' NEOAP            CPU1 010 NEOJOB   05/10/08 00.01 00.00.01 YN   0   0 C NN
****************************** Bottom of data ******************************
```

Entering the L row command as in Example 2-10 results in this message:

```
09/29 12.10.32 EQQM923I JOBLOG FOR NEOJOB   (JOB05878) ARRIVED
CN(INTERNAL)
```

Diagnose DataStore

Press Enter to see the sysout displayed.

If entering the L row command the second time results in an error message, begin troubleshooting DataStore:

- Is there a sysout with the destination you supplied on the parms for DataStore and the Controller on the JES SPOOL?
- Is RCLEANUP(YES) in the Controller parms?
- Is RCLOPTS DSTDEST(TWS82DST) in the Controller parms and is it the same as the DataStore destination?
- Is the FLOPTS set up properly? (Refer to Chapter 5, “Initialization statements and parameters” on page 97.)
- Are XCFOPT and Route opt set up properly in the Controller parms?
- Check the DataStore parameters DSTOPTS, Sysdest to make sure it is the same as the Controller DSTDEST. Check to make sure the DSTGROUP, DSTMEM, and CTLMEM are set up properly.

Important: Make sure that there is not a sysout archiver product archiving and deleting the sysout before DataStore gets the check to pick it up.

To finish the checkout refer to Chapter 8, “Tivoli Workload Scheduler for z/OS Restart and Cleanup” on page 181, and follow the procedure to restart a job.

3

The started tasks

Scheduling on multiple platforms involves controlling jobs from a central point and submitting jobs to one of several systems. Learning what each started task accomplishes gives a better understanding of how the job might flow from one system to another, how the status of that job is tracked, and how the status is reported back to the scheduler.

We discuss the started tasks and their function, and give a simple example of how to configure them. We describe how a job is submitted and tracked, and the use of the APPC and TCP/IP Server.

This chapter includes:

- Overview
- The Controller started task
- The Tracker started task
- The DataStore started task
- Connecting the primary started tasks
- The APPC Server started task

3.1 Overview

The purpose of the chapter is to give an overall understanding of how the started tasks work together to accomplish the task of scheduling a job. We introduce the functions of the started tasks and how each of them is configured. We cover how a job is submitted and tracked, and status is reported back to the scheduler. We show the procedures of each of the started tasks and define the data set of each. And we discuss performance impacts of certain data sets.

3.2 The Controller started task

The Controller, as the name implies, is the control for Tivoli Workload Scheduler for z/OS scheduling, receiving and transmitting information from and to the other started tasks using this information to control the schedule. It communicates with the other started tasks using XCF, VTAM, or a Shared DASD device. (The shared DASD is the slowest.) Refer to Chapter 4, “Tivoli Workload Scheduler for z/OS communication” on page 87.

Some of the functions of the Controller are:

- Submit jobs to the current plan
- Restart jobs
- Monitor jobs in the current plan
- Auto restart jobs
- Monitor special resources and event triggers
- Display recalled sysout
- Update databases, such as the Application Description database
- Communicate with the Tracker and DataStore
- Transmit JCL to the Tracker when a job is submitted

3.2.1 Controller subtasks

The Controller subtasks are described in detail in *IBM Tivoli Workload Scheduler for z/OS Diagnosis Guide and Reference Version 8.2,* SC32-1261. To display the status of these subtasks use the following MVS™ command:

```
F cnt1,status,subtask
```

cnt1 indicates the Started task name.

Controller subtask definitions

Controller subtask definitions are:

- The Normal Mode Manager (NMM) subtask manages the current plan and long-term plan related data sets.
- The Event Manager (EMGR) subtask processes job-tracking and user-created events and updates the current plan accordingly.
- The Job-tracking Log Archiver (JLA) subtask asynchronously copies the contents of the inactive job-tracking data set to the JT archive data set.
- The External Router (EXA) subtask receives submit requests from the data router subtask when an operation is ready to be started at a computer workstation that specifies a user-defined destination ID.
- The Workstation Analyzer (WSA) subtask analyzes operations (jobs, started tasks, and WTO messages) that are ready to start.
- The General Service (GEN) subtask services a queue of requests from the dialogs, batch loader, and program interface to the Controller. General Service Executors process the requests that are on the GS queue. The GS task can attach up to five GS executor tasks to prevent service requests from being queued.
- The Automatic Recovery (AR) subtask handles automatic recovery requests.
- The TCP/IP Tracker Router (TA) subtask is responsible for all communication with TCP/IP-connected Tracker agents.
- The APPC Tracker Router (APPC) subtask is responsible for all communication with APPC-connected Tracker agents.
- The End-to-End (TWS) subtask handles events to and from fault-tolerant workstations (using the Tivoli Workload Scheduler for z/OS TCP/IP Server).
- The Fetch job log (FL) subtask retrieves JES JOBLOG information.
- The Pre-SUBMIT Tailoring (PSU) subtask, used by the restart and cleanup function, tailors the JCL before submitting it by adding the EQQCLEAN pre-step.

3.2.2 Controller started task procedure

The EQQJOBS CLIST that is run during installation builds tailored started task procedures and jobs to allocate needed data sets (described in Chapter 1, “Tivoli Workload Scheduler for z/OS installation” on page 3, and in *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264). These data sets must be sized based on the number of jobs in the database, using the guidelines referenced in this chapter. Before starting these started task procedures, it is important to edit them and make sure they are correct and contain the HLQ

names that you need. Example 3-1 shows the Controller procedure. Note that the parm TWSC is the name of the TWSparmlibmember. EQQCONO in the EQQJOBS Install data set is the member that contains the Controller procedure.

Example 3-1 Controller procedure

```
//TWSC PROC
//STARTING EXEC TWSC
//TWSC       EXEC PGM=EQQMAJOR,REGION=0M,PARM='TWSC',TIME=1440
//STEPLIB   DD  DISP=SHR,DSN=TWS.INST.LOADLIB
//EQQMLIB   DD  DISP=SHR,DSN=EQQ.SEQQMSG0
//EQQMLOG   DD  DISP=SHR,DSN=TWS.CNTLR.MLOG
//EQQPARM   DD  DISP=SHR,DSN=TWS.INST.PARM
//SYSMDUMP  DD  SYSOUT=*
//EQQDUMP   DD  SYSOUT=*
//EQQBRDS   DD  SYSOUT=(A,INTRDR)
//EQQEVDS   DD  DISP=SHR,DSN=TWS.INST.TWSC.EV
//EQQCKPT   DD  DISP=SHR,DSN=TWS.INST.TWSC.CKPT
//EQQWSDS   DD  DISP=SHR,DSN=TWS.INST.TWSC.WS
//EQQADDS   DD  DISP=SHR,DSN=TWS.INST.TWSC.AD
//EQQRDDS   DD  DISP=SHR,DSN=TWS.INST.TWSC.RD
//EQQSIDS   DD  DISP=SHR,DSN=TWS.INST.TWSC.SI
//EQQLTDS   DD  DISP=SHR,DSN=TWS.INST.TWSC.LT
//EQQJS1DS  DD  DISP=SHR,DSN=TWS.INST.TWSC.JS1
//EQQJS2DS  DD  DISP=SHR,DSN=TWS.INST.TWSC.JS2
//EQQOIDS   DD  DISP=SHR,DSN=TWS.INST.TWSC.OI
//EQQCP1DS  DD  DISP=SHR,DSN=TWS.INST.TWSC.CP1
//EQQCP2DS  DD  DISP=SHR,DSN=TWS.INST.TWSC.CP2
//EQQNCPDS  DD  DISP=SHR,DSN=TWS.INST.TWSC.NCP
//EQQCXDS   DD  DISP=SHR,DSN=TWS.INST.TWSC.CX
//EQQNCXDS  DD  DISP=SHR,DSN=TWS.INST.TWSC.NCX
//EQQJTARC  DD  DISP=SHR,DSN=TWS.INST.TWSC.JTARC
//EQQJTO1   DD  DISP=SHR,DSN=TWS.INST.TWSC.JT1
//EQQJTO2   DD  DISP=SHR,DSN=TWS.INST.TWSC.JT2
//EQQJTO3   DD  DISP=SHR,DSN=TWS.INST.TWSC.JT3
//EQQJTO4   DD  DISP=SHR,DSN=TWS.INST.TWSC.JT4
//EQQJTO5   DD  DISP=SHR,DSN=TWS.INST.TWSC.JT5
//EQQJCLIB  DD  DISP=SHR,DSN=TWS.INST.JCLIB
//EQQINCWK  DD  DISP=SHR,DSN=TWS.INST.INCWORK
//EQQSTC    DD  DISP=SHR,DSN=TWS.INST.STC
//EQQJBLIB  DD  DISP=SHR,DSN=TWS.INST.JOBLIB
//EQQPRLIB  DD  DISP=SHR,DSN=TWS.INST.JOBLIB
//*   datasets FOR DATA STORE
//EQQPKI01  DD  DISP=SHR,DSN=TWS.INST.PKI01
//EQQSKI01  DD  DISP=SHR,DSN=TWS.INST.SKI01
```

```
//EQQSDF01  DD  DISP=SHR,DSN=TWS.INST.SDF01
//EQQSDF02  DD  DISP=SHR,DSN=TWS.INST.SDF02
/EQQSDF03  DD  DISP=SHR,DSN=TWS.INST.SDF03
```

Table 3-1 shows each of the Controller data sets and gives a short definition of each.

Table 3-1 Controller data sets

DD name	Description
EQQEVxx	Event dataset
EQQMLIB	Message library
EQQMLOG	Message logging dataset
EQQPARM	TWS parameter library
EQQDUMP	Dump dataset
EQQMDUMP	Dump dataset
EQQBRDS	Internal Reader definition
EQQCKPT	Checkpoint dataset
EQQWSDS	Workstation/Calendar depository
EQQADDS	Application depository
EQQRDDS	Special Resource depository
EQQSIDS	ETT configuration information
EQQLTDS	Long-term plan dataset
EQQJSx	JCL repository
EQQOIDS	Operator information dataset
EQQCP1DS	Current plan dataset
EQQCP2DS	Current plan dataset
EQQNCPDS	New current plan dataset
EQQNCXDS	New current plan extension dataset
EQQJTARC	Job track archive
EQQJTxx	Job track log
EQQJCLIB	JCC message table

DD name	Description
EQQINCWK	JCC incident work file
EQQSTC	Started Task Submit file
EQQJBLIB	Job Library
EQQPRLIB	Automatic Recovery Library
EQQPKI01	Local DataStore Primary Key Index file
EQQSKI01	Local DataStore Structured Key Index file
EQQSDFxx	Local DataStore Structured Data file

Important: The EQQCKPT data set is very busy. It should be placed carefully on a volume with not much activity.

3.3 The Tracker started task

The function of the Tracker is to track events on the system and send those events back to the Controller, so the current plan can be updated as these events are happening on the system. This includes tracking the starting and stopping of jobs, triggering applications by detecting a close of an update to a data set, a change in status of a special resource, and so forth. The Tracker will also submit jobs using the JES internal reader.

3.3.1 The Event data set

The Event data set is used to track events that are happening on the system. The job events are noted below as well as the triggering events. The first byte in an exit record is A if the event is created on a JES2 system, or B if the event is created on a JES3 system. This byte is found in position 21 of a standard event record, or position 47 of a continuation (type N) event. Bytes 2 and 3 in the exit record define the event types. These event types are generated by Tivoli Workload Scheduler for z/OS for jobs and started tasks (shown as a JES2 system), as shown in Example 3-2.

Example 3-2 Events for jobs and resources

```
A1 Reader event. A job has entered the JES system
A2 Job-start event. A job has started to execute.
AS Step-end event.A job step has finished executing.
A3J Job-end event. A job has finished executing.
Aup Job-termination event. A job has been added to the JES output queues.
```

```
A4 Print event. An output group has been printed.
A5 Purge event. All output for a job has been purged from the JES system.
SYY Resource event. A Resource event has occurred.
```

If you are missing events in the Event data set (EQQEVDS), the Tracker is not tracking properly and will not report proper status to the Controller. To resolve this, search the Event data set for the job submitted and look for the missing event. These events originate from the SMF/JES exit, so most likely the exit is not working correctly or may not be active. *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264, contains a chart that correlates events with exits. Also the *IBM Tivoli Workload Scheduler for z/OS Diagnosis Guide and Reference Version 8.2,* SC32-1261, has a layout of the event records.

Tracker subtask definitions

The Tracker subtask definitions are:

EWTR The Event Writer subtask writes event records to an event data set.

JCC The Job Completion Checker subtask provides support for job-specific and general checking of SYSOUT data sets for jobs entering the JES output queues.

VTAM The Network Communication Function subtask supports the transmission of data between the controlling system and controlled systems connected via VTAM/SNA.

SUB The Submit subtask supports job submit, job release, and started-subtask initiation.

ERDR The Event Reader subtask provides support for reading event records from an event data set.

DRT The Data Router subtask supports the routing of data between Tivoli Workload Scheduler for z/OS subtasks. Those subtasks may run within the same component or not.

RODM This subtask supports use of the Resource Object Data Manager (RODM) to track the status of real resources used by Tivoli Workload Scheduler for z/OS operations.

APPC The APPC/MVS subtask facilitates connection to programs running on any Systems Application Architecture® (SAA®) platform, and other platforms that support Advanced Program-to-Program Communication (APPC).

3.3.2 The Tracker procedure

The EQQJOBS CLIST that runs during the installation builds tailored started task procedures. This is described in Chapter 1, “Tivoli Workload Scheduler for z/OS installation” on page 3 and *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264. Before starting these procedures, it is important to edit them to be sure they are correct and contain the HLQ names you need.

Example 3-3 shows the Tracker procedure. Note that the parm TWST is the started task name of the Tracker.

Example 3-3 Tracker procedure

```
//TWST  PROC
//TWST      EXEC PGM=EQQMAJOR,REGION=7M,PARM='TWST',TIME=1440
//STEPLIB   DD  DISP=SHR,DSN=EQQ.SEQQLMDO
//EQQMLIB   DD  DISP=SHR,DSN=EQQ.SEQQMSGO
//EQQMLOG   DD  SYSOUT=*
//EQQPARM   DD  DISP=SHR,DSN=TWS.INST.PARM
//SYSMDUMP  DD  SYSOUT=*
//EQQDUMP   DD  SYSOUT=*
//EQQBRDS   DD  SYSOUT=(A,INTRDR)
//EQQEVDO1  DD  DISP=SHR,DSN=TWS.INST1.EV
//EQQJCLIB  DD  DISP=SHR,DSN=TWS.INST.JCLIB
//EQQINCWK  DD  DISP=SHR,DSN=TWS.INST.INCWORK
```

Table 3-2 shows descriptions of the Tracker DD statements in the procedure.

Table 3-2 Tracker DD statements in the procedure

DD statement	Description
EQQMLIB	Message library
EQQMLOG	Message logging file
EQQPARM	Parameter file for TWS
SYSMDUMP	Dump dataset
EQQDUMP	Dump dataset
EQQ BRDS	Internal Reader
EQQEVxx	Event dataset (must be unique to the Tracker)
EQQJCLIB	JCC incident library
EQQINCWK	JCC incident work file

3.3.3 Tracker performance

Tracker performance can be affected if the event data set is not placed properly. The event data set is very busy, so it should be placed on a volume with limited activity. If the event data set is slowed down by other I/O activity, the status of jobs completing will slow considerably. In the event that a volume with this data set is locked up (for example, in a full volume backup) it may affect the tracking considerably.

Another factor that may affect performance of the Tracker is JES. Because some of the events come from the JES exits, if JES is slow in reporting a job finishing, or a job purging the print, the response the Scheduler sees on the Controller will be slow also.

Important: The Tracker must be non-swappable (see 1.3.8, "Updating SCHEDxx member" on page 11), and must have the same priority as JES.

3.4 The DataStore started task

The DataStore started task captures sysout from the JES spool when a job completes. It does this by looking at the JES data sets and determining the Destination that is set up in the DataStore parms.This sysout is requested by the Controller when restarting a job or when the L row command (get listing) is used to recall the sysout for display purposes.

When the job is submitted, two JCL cards are inserted by the Controller that give JES the command to create a sysout data set and queue as a destination. DataStore looks for this Destination, reads and stores the sysout to the DataStore database, then deletes it in JES. This Destination is set up in the Controller parms and must be the same as the destination in the DataStore parms. Example 3-4 shows the JCL that is inserted at the beginning of every job.

Example 3-4 JCL inserted by the Controller

```
//TIVDSTOO OUTPUT JESDS=ALL,DEST=OPC
//TIVDSTAL OUTPUT JESDS=ALL
```

3.4.1 DataStore procedure

DataStore uses the EQQPKI, EQQSDF, and EQQUDF as the database to store the sysout. EQQPARM is used to set up the parms for DataStore, including the cleanup of the database, the configuration parms, and the destination parms (Example 3-5 on page 78).

Example 3-5 DataStore procedure

```
//TWSD PROC
//TWSDST    EXEC PGM=EQQFARCH,REGION=0M,PARM='EQQDSTP',TIME=1440
//STEPLIB   DD  DISP=SHR,DSN=EQQ.SEQQLMD0
//EQQMLIB   DD  DISP=SHR,DSN=EQQ.SEQQMSG0
//EQQPARM   DD  DISP=SHR,DSN=TWS.INST.PARM(DSTP)
//EQQPKI01  DD  DISP=SHR,DSN=TWS.INST.DS.PKI01
//EQQSKI01  DD  DISP=SHR,DSN=TWS.INST.DS.SKI01
//EQQSDF01  DD  DISP=SHR,DSN=TWS.INST.DS.SDF01
//EQQSDF02  DD  DISP=SHR,DSN=TWS.INST.DS.SDF02
//EQQSDF03  DD  DISP=SHR,DSN=TWS.INST.DS.SDF03
//EQQUDF01  DD  DISP=SHR,DSN=TWS.INST.DS.UDF01
//EQQUDF02  DD  DISP=SHR,DSN=TWS.INST.DS.UDF02
//EQQUDF03  DD  DISP=SHR,DSN=TWS.INST.DS.UDF03
//EQQUDF04  DD  DISP=SHR,DSN=TWS.INST.DS.UDF04
//EQQUDF05  DD  DISP=SHR,DSN=TWS.INST.DS.UDF05
//EQQMLOG   DD  SYSOUT=*
//EQQDUMP   DD  SYSOUT=*
//EQQDMSG   DD  SYSOUT=*
//SYSABEND  DD  SYSOUT=*
```

Table 3-3 shows the descriptions of the DataStore data sets.

Table 3-3 DataStore data sets

DD name	Description
EQQMLIB	Message library
EQQPARM	TWS parameter library
EQQPKIO	Primary key index file
EQQSKIO	Structured key index file
EQQSDF	Structured Data file
EQQUDF	Unstructured data file
EQQMLOG	Message logging file

3.4.2 DataStore subtasks

Figure 3-1 illustrates the DataStore subtasks.

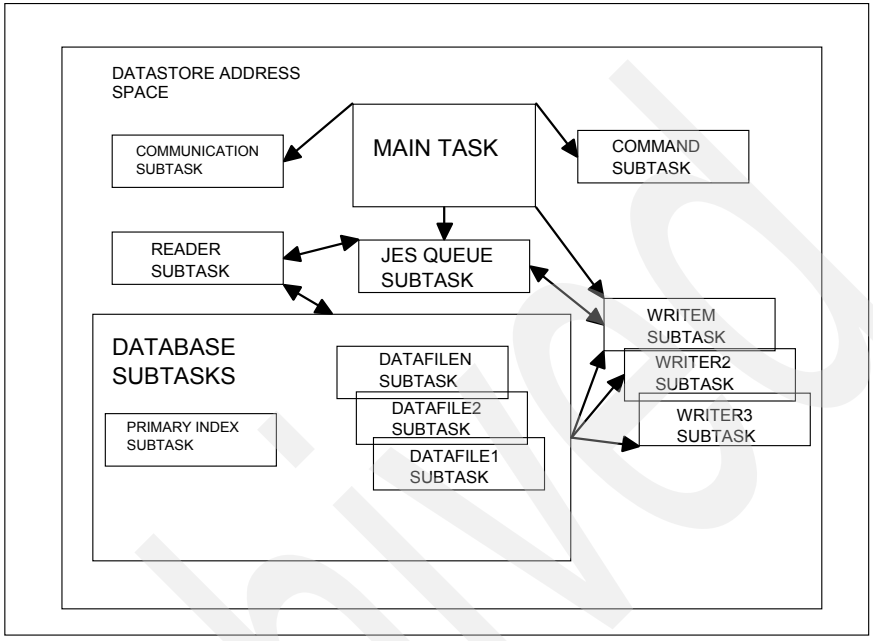


Figure 3-1 DataStore subtasks

3.5 Connecting the primary started tasks

When using a Controller and submitting jobs to the Tracker using DataStore to collect the sysout, you must connect these started tasks with XCF or VTAM (XCF being the faster of the two).

If you need to submit jobs to a second system (not sharing JES Spool) using the same Controller, you must add a Tracker and a DataStore started task, and connect them with XCF/VTAM to the Controller. In all MAS (multi-access spool) spools, one DataStore is required per MAS.

When sending a job to a Tracker, the workstation definition must be set up to point to the Tracker, and the selected workstation must be set up in the operation definition in the Application Database.

When the Controller submits the job, it pulls the JCL from JBLIB data set (JS data set on restarts), inserts the additional two JES DD statements, and transmits the JCL to the Tracker, who submits the job through the internal reader.

As the job is submitted by the Tracker, the Tracker also tracks the events (such as job starting, job ending, or job purging created by the JES/SMF exits). The Tracker writes each of these events as they occur to the Event data set and sends the event back to the Controller, which updates the status of the job on the current plan. This is reflected as updated status in the Scheduler's ISPF panels.

Example 3-6 shows started tasks working together, and Figure 3-2 on page 81 shows the parameters.

Example 3-6 Controller/DataStore/Tracker XCF

Controller

```
FLOPTS    DSTGROUP(OPCDSG)
          CTLMEM(CNTMEM)
          XCFDEST(TRKMEMA.DSTMEMA,TRKMEMB.DSTMEMB,********.********)
ROUTOPTS   XCF(TRKMEMA,TRKMEMB)
XCFOPTS   GROUP(OPCCNT)
          MEMBER(CNTMEM)
```

Tracker A

```
TRROPTS HOSTCON(XCF)
XCFOPTS GROUP(OPCCNT)
        MEMBER(TRKMEMA)
```

Tracker B

```
TRROPTS HOSTCON(XCF)
XCFOPTS GROUP(OPCCNT)
        MEMBER(TRKMEMB)
```

Data Store A

```
DSTGROUP(OPCDSG)
        DSTMEM(DSTMEMA)
        CTLMEM(CNTMEM)
```

Data Store B

```
DSTGROUP(OPCDSG1)
        DSTMEM(DSTMEMB)
        CTLMEM(CNTMEM)
```

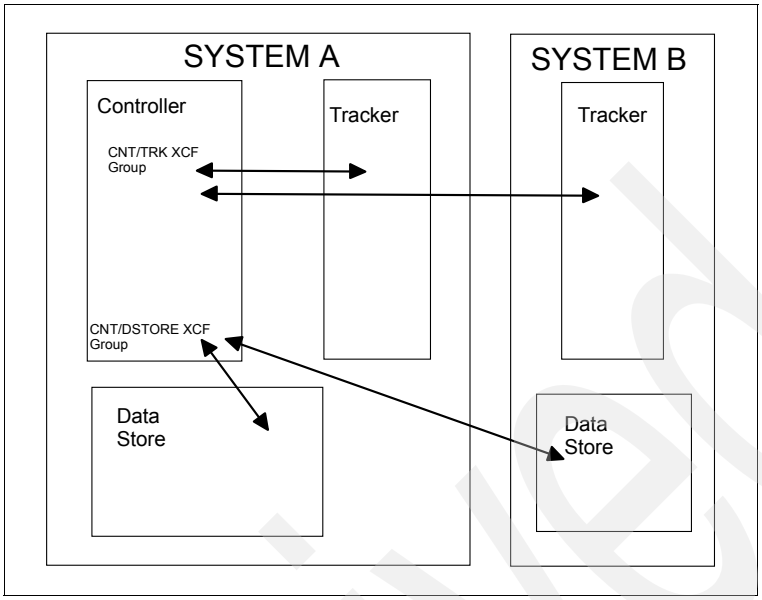


Figure 3-2 Multi-system non-shared JES spool

When starting TWS, the started task should be started in the following order:

1. Tracker
2. DataStore
3. Controller
4. Server

Important: When stopping Tivoli Workload Scheduler, do so in the reverse order and use the stop command not the cancel command.

3.6 The APPC Server started task

The purpose of the APPC Server task is to be able to access Tivoli Workload Scheduler for z/OS from a remote system. APPC is required on all systems that will use the APPC Server, as well as the system where the Controller is running. The APPC Server task must run on the system with the Controller. ISPF panels and Tivoli Workload Scheduler for z/OS data sets are required on the remote system.

To use the remote system and access Tivoli Workload Scheduler for z/OS you must access the Tivoli Workload Scheduler for z/OS ISPF panels, and set up the connection with APPC Server,

1. To configure the TWS option from the primary panel, enter `=0.1` on the command line (Figure 3-3).

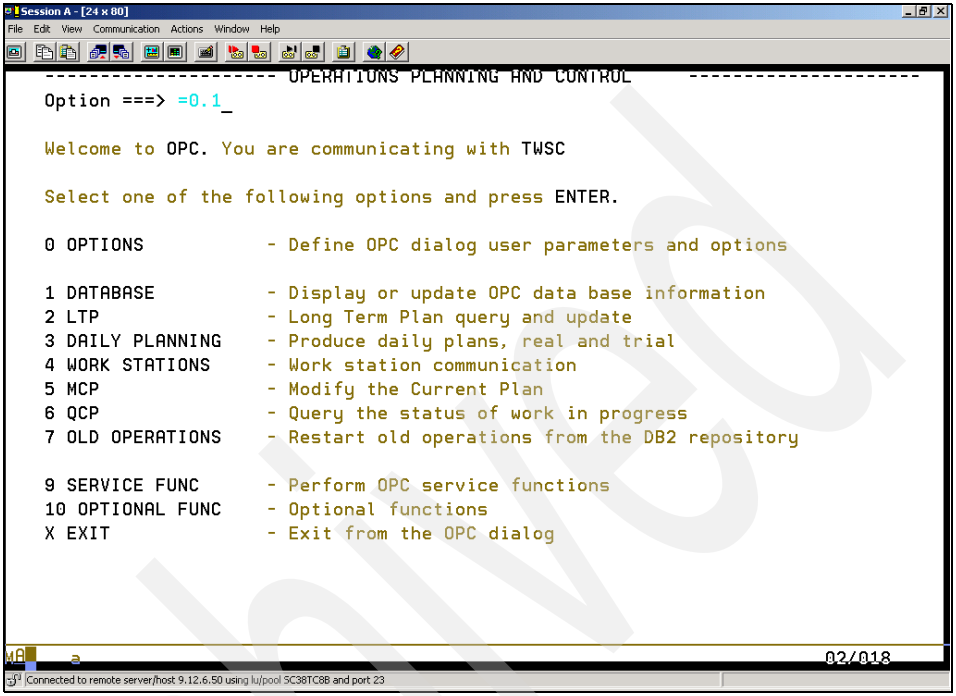


Figure 3-3 Getting to the Options panel

2. Enter Controller Started Task name, LU of appc, and a description as in Figure 3-4.

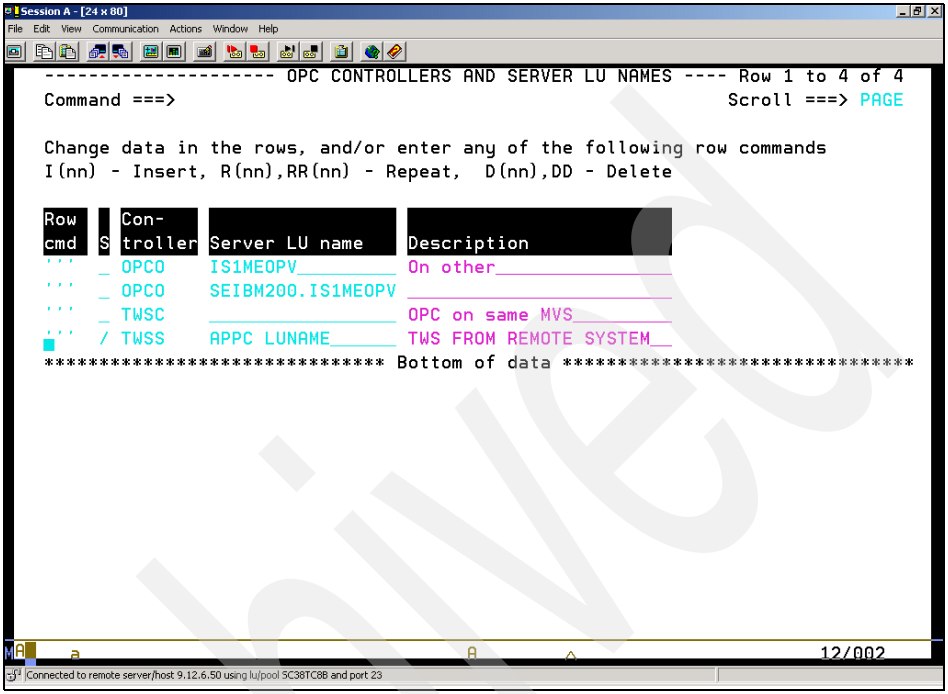


Figure 3-4 Setting up the Server options

3. Press PF3 and you will be back at the primary Tivoli Workload Scheduler for z/OS panel. Here you can access the Tivoli Workload Scheduler for z/OS Controller and its database.

3.6.1 APPC Server procedure

The EQQMLIB is the load library, and EQQMLOG is for message logging. The EQQMLOG on the server produces limited messages, so you could direct this dd to sysout=* and save dasd space. EQQPARM is the parmlib member for the APPC Server. Example 3-7 shows the procedure for the server.

Example 3-7 APPC Server started task

```
******************************** Top of Data ****************************
//TWC1S    EXEC PGM=EQQSERVR,REGION=6M,TIME=1440
//EQQMLIB   DD  DISP=SHR,DSN=TWS.V8R2M0.SEQQMSG0
//EQQMLOG   DD  SYSOUT=*
//EQQPARM   DD  DISP=SHR,DSN=EQQUSER.TWS01.PARM(SERP)
//SYSMDUMP  DD  SYSOUT=*
//EQQDUMP   DD  SYSOUT=*
//*
```

Figure 3-5 suggests what a configured APPC Server started task might look like.

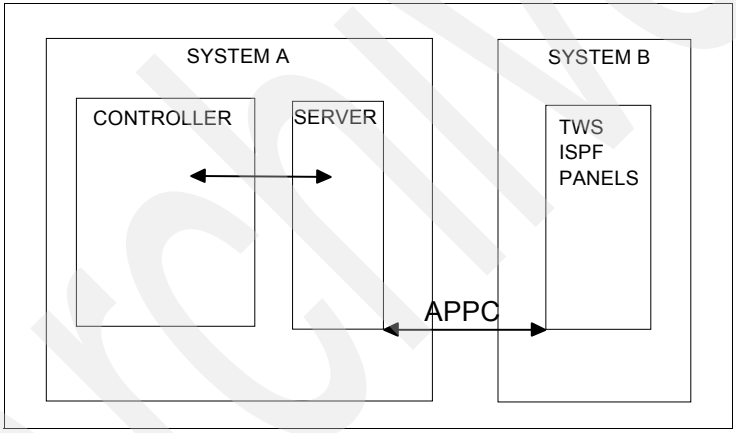


Figure 3-5 APPC Server

Example 3-8 shows examples for the APPC Server parameters.

Example 3-8 APPC Server parameters

```
/*********************************************************************/
/* SERVOPTS: run-time options for SERVER KER processor               */
/*********************************************************************/
SERVOPTS    SUBSYS(TC82) SCHEDULER(CNT1)
```

3.7 TCP/IP Server

The TCP/IP Server is used for end-to-end processing. It communicates between the Controller and the end-to-end domains or the Job Scheduling Console (a PC-based piece of software that can control both master domains and subdomains). The TCP/IP Server procedure is similar to the APPC Server, as shown in Example 3-9 (for more information see 15.1.5, “Create started task procedures” on page 393).

Example 3-9 TCP/IP procedure

```
******************************** Top of Data ***************************
//TWC1S    EXEC PGM=EQQSERVR,REGION=6M,TIME=1440
//EQQMLIB   DD  DISP=SHR,DSN=TWS.V8R2M0.SEQQMSG0
//EQQMLOG   DD  SYSOUT=*
//EQQPARM   DD  DISP=SHR,DSN=EQQUSER.TWS01.PARM(SERP)
//SYSMDUMP  DD  SYSOUT=*
//EQQDUMP   DD  SYSOUT=*
//*
```

4

Tivoli Workload Scheduler for z/OS communication

Tivoli Workload Scheduler for z/OS, having multiple started tasks, sometimes across multiple platforms, has a need to pass information between these started tasks. The protocol that is used and how it is configured may affect or speed the performance of the job scheduler. This chapter discusses these mechanisms and how to deploy them, as well as the pitfalls of the different protocols and the performance of each.

Scheduling requires submitting the job and tracking it. The Tracker keeps track of the job and has to pass this information back to the Controller. The DataStore also has to pass sysout information back to the Controller for restart purposes so the operators can see the status of a job or the job itself to see what might have gone wrong in an error condition. This passing of information is the reason that a communication mechanism is required.

This chapter includes five different communication methods and where each might be used. We cover performance impacts, ease of configuring, and hardware prerequisites.

The following topics are covered:

- Which communication to select
- XCF and how to configure it

- VTAM: its uses and how to configure it
- Shared DASD and how to configure it
- TCP/IP and its uses
- APPC

4.1 Which communication to select

When you choose a communication method for Tivoli Workload Scheduler for z/OS, you need to consider performance and hardware capability.

Tivoli Workload Scheduler for z/OS supports these sysplex (XCF) configurations:

- MULTISYSTEM - XCF services are available to Tivoli Workload Scheduler for z/OS started tasks residing on different z/OS systems.
- MONOPLEX - XCF services are available only to Tivoli Workload Scheduler for z/OS started tasks residing on a single z/OS system.

> **Note:** Because Tivoli Workload Scheduler for z/OS uses XCF signaling services, group services, and status monitoring services with permanent status recording, a couple data set is required. Tivoli Workload Scheduler for z/OS does not support a local sysplex.

In terms of performance, *XCF is the fastest option* and is the preferred method of communicating between started tasks if you have the hardware. It is easier to set up and can be done temporarily with an MVS command and permanently by setting up the parameters in SYS1.PARMLIB. You also must configure parameters in the Tivoli Workload Scheduler for z/OS parmlib.

VTAM is also an option that should be considered. VTAM is faster than the third option, shared DASD. VTAM is a little more difficult to set up, but it is still a good option because of the speed with which it communicates between started tasks.

Shared DASD also has its place, but it is the slowest of the alternatives.

4.2 XCF and how to configure it

With XCF communication links, the Tivoli Workload Scheduler for z/OS Controller can submit workload and control information to Trackers and DataStores that use XCF signaling services. The Trackers and DataStore use XCF services to transmit events to the Controller. Tivoli Workload Scheduler for z/OS systems are either ACTIVE, FAILED, or NOT-DEFINED for the Tivoli Workload Scheduler for z/OS XCF complex. Each active member tracks the state of all other members in the group. If a Tivoli Workload Scheduler for z/OS group member becomes active, stops, or terminates abnormally, the other active members are notified. Note that when using DataStore, two separate groups are required: one for the DataStore to Controller function, and a separate group for the Tracker to Controller functions.

4.2.1 Initialization statements used for XCF

Tivoli Workload Scheduler for z/OS started tasks use these initialization statements for XCF Controller/Tracker/DataStore connections:

XCFOPTS Identifies the XCF group and member name for the Tivoli Workload Scheduler for z/OS started task. Include XCFOPTS for each Tivoli Workload Scheduler for z/OS started task that should join an XCF group.

ROUTOPTS Identifies all XCF destinations to the Controller or standby Controller. Specify ROUTOPTS for each Controller and Standby Controller.

TRROPTS Identifies the Controller for a Tracker. TRROPTS is required for each Tracker on a controlled system. On a controlling system, TRROPTS is not required if the Tracker and the Controller are started in the same address space, or if they use shared DASD for event communication. Otherwise, specify TRROPTS.

Tivoli Workload Scheduler for z/OS started tasks use these initialization statements for XCF for Controller/DataStore connections:

CTLMEM Defines the XCF member name identifying the Controller in the XCF connection between Controller and DataStore.

DSTGROUP Defines the XCF group name identifying the DataStore in the XCF connection with the Controller.

DSTMEM XCF member name, identifying the DataStore in the XCF connection between Controller and DataStore.

DSTOPTS Defines the runtime options for the DataStore.

FLOPTS Defines the options for Fetch Job Log (FL) task.

XCFDEST Used by the FL (Fetch Job Log) task to decide from which DataStore the Job Log will be retrieved.

Figure 4-1 on page 91 shows an example of an XCF configuration.

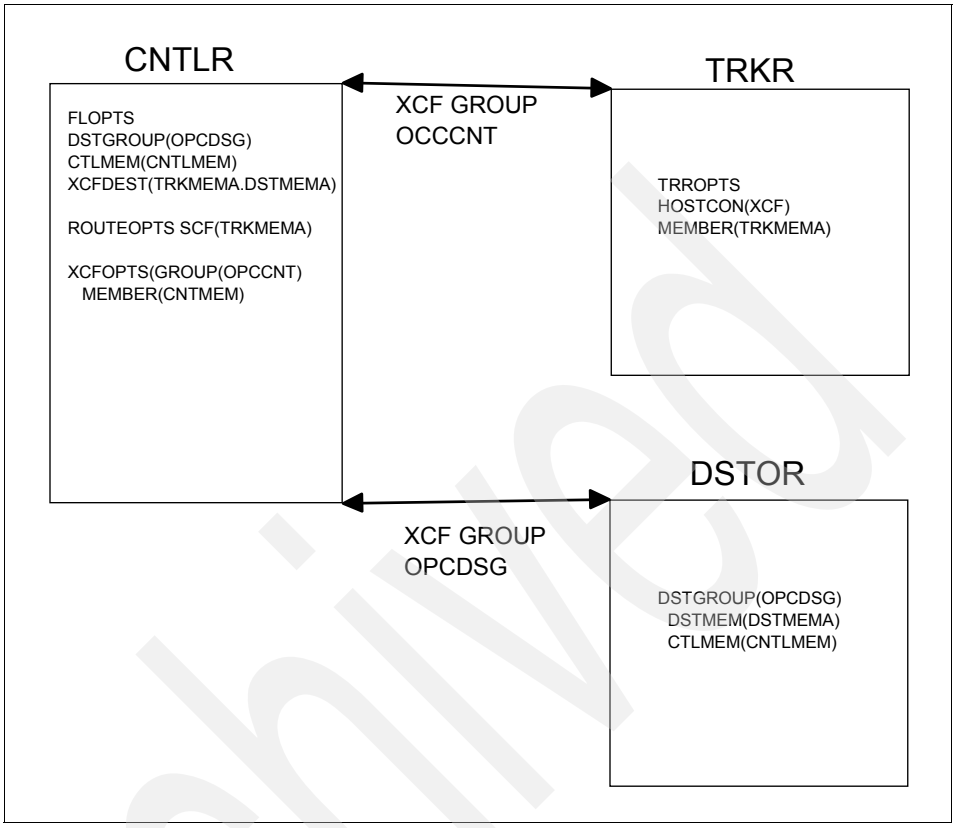


Figure 4-1 XCF example

For more details about each of the parameters, refer to *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265.

4.3 VTAM: its uses and how to configure it

Tivoli Workload Scheduler for z/OS has a subtask called Network Communication Facility (NCF), which handles the communication (using VTAM) between the Controller and Tracker. The FN task is similar and handles the communication between the Controller and DataStore. These are two separate paths that are defined as separate LUs.

You must define NCF as a VTAM application on both the controlling system and each controlled system. Before defining NCF, select names for the NCF

applications that are unique within the VTAM network.To define NCF as an application to VTAM:

- Add the NCF applications to the application node definitions, using APPL statements.
- Add the application names that NCF is known by, in any partner systems, to the cross-domain resource definitions. Use cross-domain resource (CDRSC) statements to do this.

You must do this for all systems that are linked by NCF.

For example:

- At the Controller:
  - Define the NCF Controller application. Add a VTAM APPL statement like this to the application node definitions:
    - VBUILD TYPE=APPL
    - OPCCONTR APPL VPACING=10,
    - ACBNAME=OPCCONTR
  - Define the NCF Tracker application. Add a definition like this to the cross-domain resource definitions:
    - VBUILD TYPE=CDRSC
    - OPCTRK1 CDRSC CDRM=IS1MVS2
  - Define the NCF DataStore application. Add a definition like this to the cross-domain resource definitions:
    - VBUILD TYPE=CDRSC
    - OPCDST1 CDRSC CDRM=IS1MVS2
- At the Tracker/DataStore:
  - Define the NCF Tracker application. Add a VTAM APPL statement like this to the application node definitions:
    - VBUILD TYPE=APPL
    - OPCTRK1 APPL ACBNAME=OPCTRK1,
    - MODETAB=EQQLMTAB,
    - DLOGMOD=NCFSPARM
  - Define the NCF DataStore application. Add a VTAM APPL statement like this to the application node definitions:
    - VBUILD TYPE=APPL
    - OPCDST1 APPL ACBNAME=OPCDST1,

- MODETAB=EQQLMTAB,
- DLOGMOD=NCFSPARM

– Define the NCF Controller application. Add a CDRSC statement like this to the cross-domain resource definitions:
  - VBUILD TYPE=CDRSC
  - OPCCONTR CDRSC CDRM=IS1MVS1

IS1MVS1 and IS1MVS2 are the cross-domain resource managers for the Controller and the Tracker, respectively.

Figure 4-2 shows a diagram of how the parameters might look.

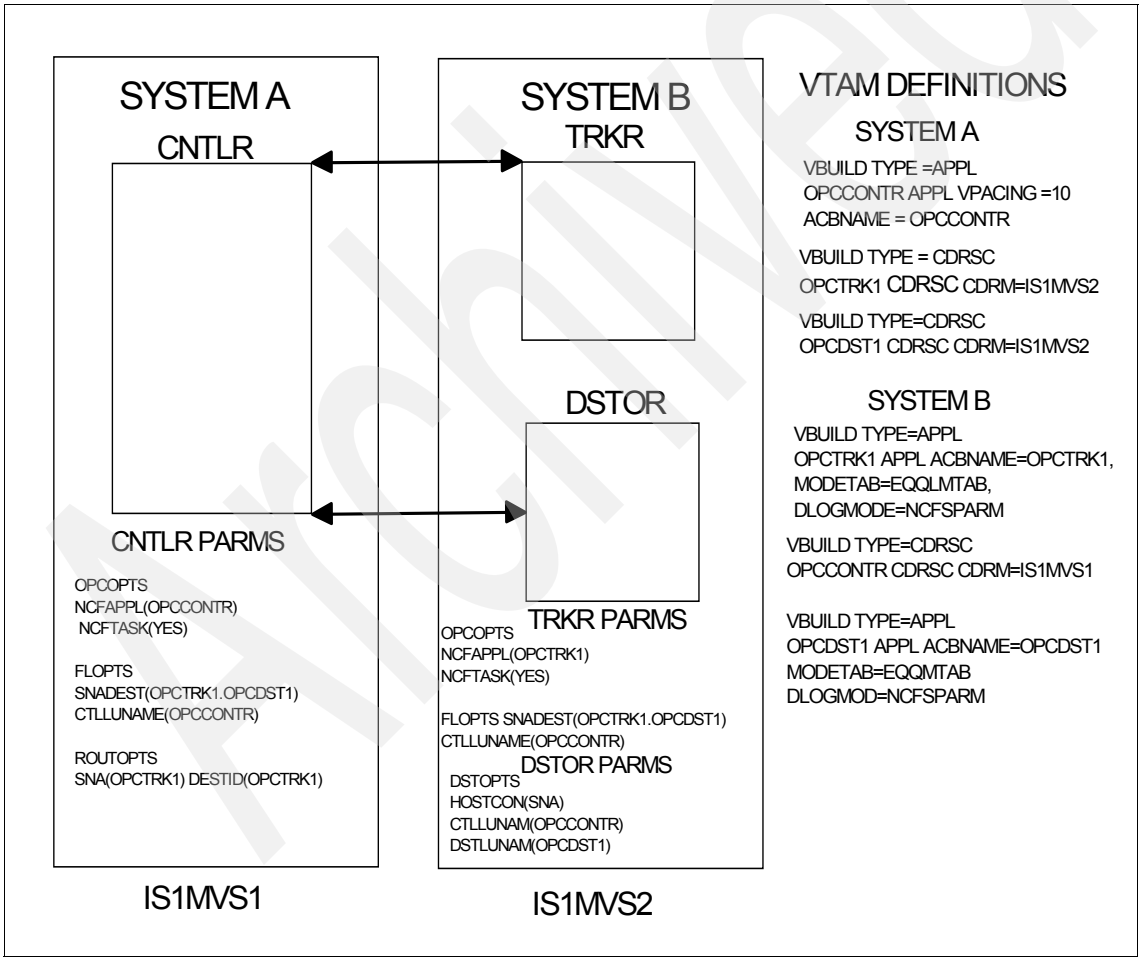


Figure 4-2 VTAM configuration and parameters

4.4 Shared DASD and how to configure it

When two Tivoli Workload Scheduler for z/OS systems are connected through shared DASD, they share two data sets for communication (Figure 4-3):

- Event data set
- Submit/release data set

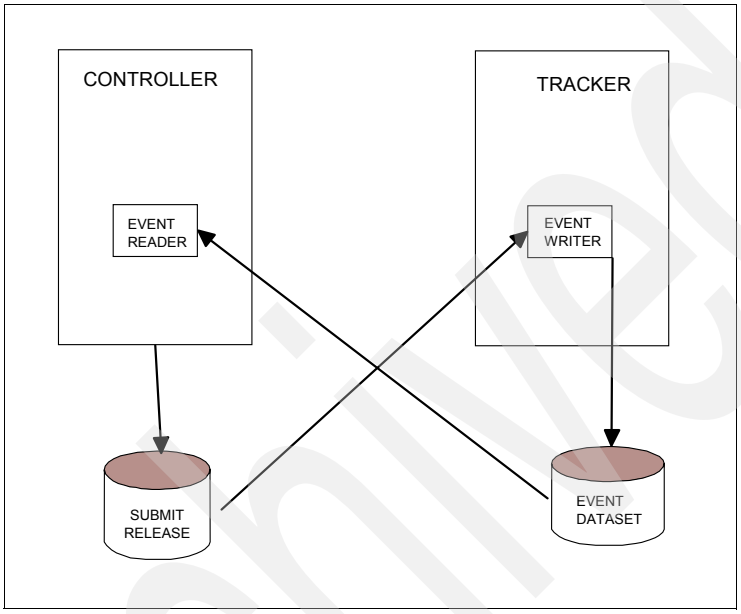


Figure 4-3 Shared DASD configuration

The Tracker writes the event information it collects to the event data set. An event reader, started in the Controller, reads the data set and adds the events to the datarouter queue. A submit/release data set is one method that the Controller uses to pass work to a controlled system. When two Tivoli Workload Scheduler for z/OS systems share a submit/release data set, the data set can contain these records:

- Release commands
- Job JCL
- Started-task JCL procedures
- Data set cleanup requests
- WTO message text

Both the host and the controlled system must have access to the submit/release data set. The EQQSUDS DD name identifies the submit/release data set in the Tracker address space. At the Controller, the DD name is user defined, but it must be the same name as that specified in the DASD keyword of the

ROUTOPTS statement. The Controller can write to any number of submit/release data sets.

You can also configure this system without a submit/release data set. When the workstation destination is blank, batch jobs, started tasks, release commands, and WTO messages are processed by the submit subtask automatically started in the Controller address space. The event-tracking process remains unchanged.

Example 4-1 shows the Tivoli Workload Scheduler for z/OS parameters for using shared DASD, as in Figure 4-3 on page 94.

Example 4-1 Shared DASD parameters for Figure 1-3

```
CONTROLLER PARMS

OPCOPTS OPCHOST(YES)
ERDRTASK(1)
ERDRPARM(STDERDR)
ROUTOPTS DASD(EQQSYSA)

TRACKER PARMS
OPCOPTS OPCHOST(NO)
ERDRTASK(0)
EWTRTASK(YES)
EWTRPARM(STDEWTR)
TRROPTS HOSTCON(DASD)

READER PARM
ERDROPTS ERSEQNO(01)

WRITER PARM
EWTROPTS SUREL(YES)
```

4.5 TCP/IP and its uses

TCP/IP is used for End to End communication to the distributed platforms. To use TCP/IP requires that you use the TCP/IP Server. This server is discussed briefly in Chapter 3, “The started tasks” on page 69.

4.6 APPC

APPC is used in Tivoli Workload Scheduler for z/OS as a mechanism to communicate between the APPC Server and the ISPF user who is logged on to a remote system. To use this function:

- APPC connections must be set up between the APPC Server and the remote user Tivoli Workload Scheduler/ISPF panels.
- APPC must be configured in Tivoli Workload Scheduler for z/OS parmlib.
- The Tivoli Workload Scheduler for z/OS ISPF Option panel must be configured.
- A Tivoli Workload Scheduler for z/OS APPC Server must be started on the same system as the Controller.

5

Initialization statements and parameters

This chapter discusses the parameters that are used to control the various features and functions of Tivoli Workload Scheduler for z/OS.

We look at the sample set of initialization statements built by the EQQJOBS installation aid, and discuss the parameter values supplied explicitly, the default values provided implicitly, and their suitability.

Systems programmers, batch schedulers, and Tivoli Workload Scheduler for z/OS administrators should gain the most from reviewing this chapter. It will help them understand how the Tivoli Workload Scheduler for z/OS functions are controlled and how it might be used, beyond its core job-ordering functions.

A full description of all the Tivoli Workload Scheduler for z/OS initialization statements and their parameters can be found in the *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265.

This chapter has the following sections:

- Parameter members built by EQQJOBS
- EQQCONOP and EQQTRAP
- EQQCONOP - STDAR
- EQQCONOP - CONOB

- EQQTRAP - TRAP
- EQQTRAP - STDEWTR
- EQQTRAP - STDJCC

5.1 Parameter members built by EQQJOBS

We start by discussing the parameter members built by EQQJOBS. (Running EQQJOBS was discussed in 1.5, “Running EQQJOBS” on page 12.)

The Tivoli Workload Scheduler for z/OS installation aid, EQQJOBS, option one, builds members into a data set that help to complete the installation. Some of these members provide sample procedures and parameter members for the Controller and Tracker.

EQQJOBS option three creates similar members to complete the DataStore installation.

EQQJOBS and all its options may be rerun as often as desired, maybe to set up a different data set-naming standard or to use different options to implement features such as End to End or Restart and Cleanup with DataStore.

Tip: It is recommended that you establish a basic working system and verify that the Controller and Trackers are communicating properly before trying to implement any of the additional Tivoli Workload Scheduler for z/OS features.

Sample parameter members are built into the install library named during the EQQJOBS process.

The most common configuration has EQQCONOP for a Tivoli Workload Scheduler for z/OS Controller started task and EQQTRAP for a Tivoli Workload Scheduler for z/OS Tracker started task running in separate address spaces.

EQQCONP provides a sample set of parameters that give the option of running the subtasks for both the Controller and Tracker in the same address space.

These members relate to the sample Controller and Tracker procedures, EQQCONOP and EQQTRAP with EQQCONO and EQQTRA, EQQCONP with EQQCON. The comments section at the top of the procedure members details any changes to be made to the procedures if you are, or are not, using certain features or configurations.

Note: The sample procedures for the Controller, Tracker, and DataStore tasks all have an EQQPARM DD statement. The library (or libraries) identified by this DD statement is where the members containing the initialization statements and parameters should be built.

EQQDST is a sample DataStore started task, and EQQDSTP is its sample parameters.

The procedures, statements, and parameters built into these members vary depending on the options chosen when running EQQJOBS. The examples below show the alternate statements and parameters used for either XCF or SNA DataStore/Controller/Tracker communication and the ones to use for End-to-End.

Tip: Three communications options are possible between the Controller and Trackers, XCF, SNA, or shared DASD. Either XCF or SNA should be used, especially if DataStore is to be used. (There is no possibility of Shared DASD communication when DataStore is involved.) Shared DASD is a very slow communications method and is to be avoided.

The parameter members EQQTRAP and EQQCONOP contain several statements, which should be split into the separate members indicated in the comments and copied into the library to be referenced by the EQQPARM DD card in the Controller, Tracker, or DataStore started task.

Example 5-1 Comments between members in the parameter samples

```
=====================================================================
====================== MEMBER STDAR  ================================
=====================================================================
```

When copying the members, note that the separator lines, as shown in the example above, will cause an error if left in the members. The syntax is invalid for a comment line.

The following discussion primarily reviews the statements found in the EQQCONOP, EQQTRAP, and EQQDSTP members, as these should cover most installations' requirements.

When starting a Tivoli Workload Scheduler for z/OS started task, Controller, Tracker, DataStore, or Server, the initialization statements and their return codes are written to that task's Message Log. The Message Log will be the data set (or more commonly sysout) identified by the EQQMLOG DD statement in the procedure for the started task.

5.2 EQQCONOP and EQQTRAP

This section lists the statements that make up EQQCONOP and EQQTRAP, split into the relevant submembers.

CONOP, the largest of the three submembers in EQQCONOP, is divided into statements for this discussion.

TRAP is the largest of the submembers in EQQTRAP.

Some statements appear in both TRAP and CONOP, but the parameters used will have values relevant to either a Tracker or a Controller task.

5.2.1 OPCOPTS from EQQCONOP

The OPCOPTS statement is valid for both Tracker and Controller subtasks. It describes how this particular Tivoli Workload Scheduler for z/OS started task should function and its purpose in life. We discuss the parameters and the values supplied by EQQJOBS and show them in the following example.

OPCHOST

This parameter tells the Tivoli Workload Scheduler for z/OS code whether this is a Controller - value YES, or a Tracker - value NO, subsystem.

Notes:

- The Controller is the subsystem that supports the dialog users and contains the databases and plans. There is only one active Controller subsystem per batch environment.
- The Tracker is the subsystem that communicates events from all of the mainframe images to the Controller. There should be one per LPAR for each Controller that runs work on that system.

There are two other values that may be used if this installation is implementing a hot standby environment: PLEX and STANDBY.

STANDBY is used when you can define the same Controller subsystem on one or more LPARs within a sysplex. It (or they) should monitor the Controller and be the first to become aware of a failure should takeover the Controller functions.

PLEX indicates that this subtask is a Controller; however, it also resides in a sysplex environment with the same Controller on other LPARs. None of the Controllers in this environment would have a value of YES. PLEX causes the first Controller that starts to become the Controller. When the other Controllers start, they wait in standby mode.

For a first installation it is advisable to use OPCHOST(YES) as shown in Example 5-2 on page 102, and to consider the use of STANDBY and PLEX in the future when the environment is stable.

Note: The Hot Standby feature of Tivoli Workload Scheduler for z/OS enables the batch scheduling environment to be moved to another LPAR, automatically, if that the databases are held on shared DASD.

Example 5-2 OPCOPTS from CONOP

```
/*********************************************************************/
/* OPCOPTS: run-time options for the CONTROLLER processor.           */
/*********************************************************************/
OPCOPTS  OPCHOST(YES)
         APPCTASK(NO)
         ERDRTASK(0)
         EWTRTASK(NO)
         GSTASK(5)
         JCCTASK(NO)
         NCFTASK(YES)       NCFAPPL(NCFCNT01)
         RECOVERY(YES)      ARPARM(STDAR)
         RODMTASK(NO)
         VARSUB(SCAN)       GTABLE(JOBCARD)
         RCLEANUP(YES)
         TPLGYSRV(OSER)
         SERVERS(OSER)
/*-------------------------------------------------------------------*/
/* If should specify the following the first time you run OPC        */
/* after the installation/migration:                                 */
/*       BUILDSSX(REBUILD)                                           */
/*       SSCMNAME(EQQSSCMF,PERMANENT)                                */
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
/* If you want to use OS/390 Automatic Restart Manager with OPC      */
/* specify:                                                          */
/*       ARM(YES)                                                    */
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
/* If you want to use OS/390 Workload Manager with OPC, supposing    */
/* that BATCHOPC uis the WLM service class assigned to a critical    */
/* job, and that the wished Policy is CONDITIONAL, you must          */
/* specify:                                                          */
/*       WLM(BATCHOPC,CONDITIONAL(20))                               */
/*-------------------------------------------------------------------*/
```

APPCTASK

This parameter tells the Tivoli Workload Scheduler for z/OS code whether it should start the APPC subtask. This enables communication to an AS/400® systems Tracker, or for a program written using the Tivoli Workload Scheduler for z/OS API. It is unlikely that initially you would need to change the sample to YES.

ERDRTASK(0)

This parameter controls how many event reader tasks, if any, are to be started. Event reader tasks are needed only when communication between the Controller and Tracker is managed via shared data sets. It is more likely that communication will be via SNA or XCF, both of which are far more efficient than using shared data sets.

Note: The event reader task reads the events written by the event writer task to the event data set. Where communication is by shared data sets, the event reader subtask resides in the Controller address space, and reads the events written to the event data set by the event writer subtask, part of the Tracker address space. Where communication is via XCF or SNA, then the event reader task is started with the event writer, in the Tracker address space.

EWTRTASK(NO)

This parameter is not needed for the Controller address space, unless both the Tracker and Controller are being started in a single address space.

Note: The event writer task writes events to the event data set. These events have been read from the area of ECSA that was addressed for the Tracker at IPL time and defined in the IEFSSNxx member when the Tracker subsystem was defined. Events written to the ECSA area by SMF and JES exits contain information about jobs being initiated, starting, ending, and being printed and purged. User events are also generated by Tivoli Workload Scheduler for z/OS programs, such as EQQEVPGM.

GSTASK(5)

The General Services Task handles requests for service from the dialog users; user-written and supplied PIF programs such as BCIT, OCL, and the JSC (the Tivoli Workload Scheduler for z/OS GUI); and Batch Loader requests. The numeric value associated with this parameter indicates how many executor tasks are to be started to manage the General Services Tasks queue. Any value between 1 and 5 may be used, so the sample value of 5 need not be changed. This parameter is valid only for a Controller.

JCCTASK(NO)

The JCC subtask is a Tracker-related parameter. You would only need it in a Controller started task, if the Controller and Tracker were sharing the same address space, and the JCC functionality was required.

If used in a Tracker, it will be joined by the JCCPARM parameter, which will identify the parameter member that will control the JCC function. See 5.2.2, “OPCOPTS from EQQTRAP” on page 106.

Note: The JCC feature reads the output from a job to determine, beyond the job’s condition code, the success or failure of the job. It scans for specific strings of data in the output and alters the completion code accordingly.

NCFTASK(YES) and NCFAPPL(luname)

These parameters tell the Controller and the Tracker that communication between them will be via SNA. A Network Communications Functions (NCF) subtask will be started. The luname defined by the NCFAPPL parameter is the one that has been defined to represent the task.

See *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264, for full details on the VTAM definitions needed.

Note: When using XCF to communicate between the Controller and Tracker tasks, this statement is not needed, and there is no equivalent statement to turn on XCF communication. See the ROUTOPTS, XCFOPTS, and TRROPTS later in this chapter to complete the discussion on Tracker and Controller communication.

RECOVERY(YES) and ARPARM(arparm)

Specifying YES causes the Automatic Recovery subtask to be started in a Controller. It will be controlled by the values defined in the parameter member indicated by the ARPARM parameter. It is unlikely that you will initially use this feature, but switching it on will not cause Tivoli Workload Scheduler for z/OS to suddenly start recovering failed jobs, and nothing will happen unless someone codes Automatic Recovery statements in their JCL.

VARSUB(SCAN) and GTABLE(JOBCARD)

The Controller can substitute variables in a job’s JCL prior to its submission onto the system. A VARSUB value of SCAN causes Tivoli Workload Scheduler for z/OS to look only for the SCAN directive in the JCL. When encountered it will action any other Tivoli Workload Scheduler for z/OS directives and attempt to resolve any variables it finds. This value is a performance improver over the value

of YES, which causes Tivoli Workload Scheduler for z/OS to look for directives and variables on every line of every job's JCL. Leaving the sample value of SCAN enables this function to be introduced without the need to change the existing Tivoli Workload Scheduler for z/OS setup.

When Tivoli Workload Scheduler for z/OS encounters a variable in the JCL, it attempts to resolve it, following all sorts of rules, but eventually it will drop down to look in the default user-defined variable table. If the default table named on the GTABLE (Global Table) parameter does not exist, it will cause an error. Ensure you create a user variable table called JOBCARD, or create one by another name, for example DEFAULT or GLOBAL, and change the GTABLE value accordingly.

See *IBM Tivoli Workload Scheduler for z/OS Managing the Workload Version 8.2,* SC32-1263, for full details about Job Tailoring.

RCLEANUP(YES)

This parameter switches on the Restart and Cleanup feature of the Tivoli Workload Scheduler for z/OS Controller. It requires the DataStore task. Initially you may wish to start Tivoli Workload Scheduler for z/OS without this feature, and add it at a later date. Even where your installations JCL standards may mean there is little use for the Restart and Cleanup function, one of the elements of this feature, joblog retrieval, may be a useful addition.

TPLGYSRV(member)

This parameter should be coded only if you want the Controller to start the End to End feature of Tivoli Workload Scheduler. The member that would be specified is the name of the server that handles communication between the Controller and the end-to-end server.

SERVERS(server1,server2,server3)

Initially you will probably not need this parameter. However, when you have decided on your final configuration, and you determine that you will need some server tasks to handle communication to the Controller from PIF programs, or you decide to use the end-to-end feature, then you can list the server task names here. The Controller will then start and stop the server tasks when it starts and stops.

BUILDSSX(REBUILD) SSCMNAME(EQQSSCMF,PERMANENT)

This parameter is commented out in the sample. It is used when migrating to a different Tivoli Workload Scheduler for z/OS release, or when the EQQINITx module has been altered by a fix. These parameters enable you to rebuild the subsystem entry for the started task using a different module. This allows for the

testing of a new release and (when permanent is used) the go-live of a new release, without an IPL.

This parameter should remain commented out.

ARM(YES)

This parameter is commented out of both the Controller and Tracker examples. If you would like the z/OS automatic restart manager to restart the started task if it fails, uncomment this line. However, you may prefer to place the Tivoli Workload Scheduler for z/OS started tasks under the control of your systems automation product. During initial set-up it is advisable to leave this parameter commented out until a stable environment has been established.

WLM(BATCHOPC,CONDITIONAL(20))

This parameter is commented out in the sample. You may use it for your Controller if your installation wants to use the late and duration information calculated in the Tivoli Workload Scheduler for z/OS plans to move specific jobs into a higher-performing service class.

Initially this parameter should remain commented out. The Tivoli Workload Scheduler for z/OS databases must be very well defined, and the plans built from them realistic, before the data will provide sufficiently accurate information to make use of this parameter valuable.

5.2.2 OPCOPTS from EQQTRAP

Example 5-3 shows the OPCOPTS statement for the Tracker task. Each of these parameters has been discussed more fully above.

OPCHOST is set to NO, as this is not the Controller or a Standby Controller. There is no ERDRTASK, as the event reader task has been started within the event writer (see 5.7, “EQQTRAP - STDEWTR” on page 146).

Communication to the Controller is via SNA as the NCFTASK has been started, and the Tracker is identified by the luname NCFTRK01.

The JCC task is also to be started.

Example 5-3 OPCOPTS from TRAP

```
/*********************************************************************/
/* OPCOPTS: run-time options for the TRACKER processor               */
/*********************************************************************/
OPCOPTS  OPCHOST(NO)
         ERDRTASK(0)
```

```
         EWTRTASK(YES)      EWTRPARM(STDEWTR)
         JCCTASK(YES)       JCCPARM(STDJCC)
         NCFTASK(YES)      NCFAPPL(NCFTRK01)
/*--------------------------------------------------------------------*/
/* If you want to use Automatic Restart manager you must specify:    */
/*       ARM(YES)                                                     */
/*--------------------------------------------------------------------*/
```

Important: When parameters such as EWTRTASK are set to YES, Tivoli Workload Scheduler for z/OS expects to find the statements and parameters that control that feature in a parmlib member. If you do not specify a member name using a parameter such as EWTRPARM(xxxx), the default member name (the one that has been shown in the examples and headings) will be used.

5.2.3 The other OPCOPTS parameters

There are more parameters than appear in the sample for the OPCOPTS statement. The rest have been tabulated here and show the default value that will be used, if there is one. More information about these parameters can be found later in this chapter.

You may find it useful to copy the sample to your Tivoli Workload Scheduler for z/OS parmlib library and update it with all the possible parameters (Table 5-1), using the default values initially, and supplying a comment for each, or adding the statement but leaving it commented out. Coding them alphabetically will match the order in the manual, enabling you to quickly spot if a parameter has been missed, or if a new release adds or removes one.

Table 5-1 OPCOPTS parameters

Parameter	Default	Function / comment
CONTROLLERTOKEN	*this subsystem*	History function
DB2SYSTEM		History function
EXIT01SZ	0	EQQUX001
EXTMON	NO	TBSM only at this time
GDGNONST	NO	To test or not to test the JCFGDG bit to ID a GDG
GMTOFFSET	0	GMT clock shows GMT
OPERHISTORY	NO	History function
RCLPASS	NO	Restart & Cleanup

Parameter	Default	Function / comment
RODMPARM	STDRODM	Special Resources and RODM
RODMTASK	NO	Special Resources and RODM
VARFAIL		Job tailoring
VARPROC	NO	Job Tailoring
SPIN	NO	JESSPIN supported

5.2.4 CONTROLLERTOKEN(ssn), OPERHISTORY(NO), and DB2SYSTEM(db2)

These parameters control the HISTORY function in the Tivoli Workload Scheduler for z/OS Controller. If OPERHISTORY defaults to or specifically sets a value of NO, then using either of the other two statements will generate an error message.

Note: The History function provides the capability to view old current plan information, including the JCL and Joblogs, for a period of time as specified in the initialization. It also allows for the rerun of “old” jobs.

If you wish to use the History function, then you must supply a DB2 subsystem name for Tivoli Workload Scheduler for z/OS to pass its old planning data to. The controllertoken value from the planning batch job is used when the old plan data is passed to DB2, and the Controller’s controllertoken value is used when data is retrieved from the DB2 system.

See the following guides to find out more about defining Tivoli Workload Scheduler for z/OS DB2 tables, and about the History function generally and how to use it.

- *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264
- *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265
- *IBM Tivoli Workload Scheduler for z/OS Managing the Workload Version 8.2,* SC32-1263

Tip: The BATCHOPT statement discussed a little later in this chapter also specifies parameters to do with the History function.

EXIT01SZ(0)

By default the EQQUX001 Tivoli Workload Scheduler for z/OS exit may only alter the JCL during submission; it may not add any lines to it. Using this parameter, you can specify a maximum size that the JCL may be extended to in lines. It is normal to allow this parameter to default to 0 unless you are using EQQUX001 to insert some JCL statements into jobs on submission.

For more information about Tivoli Workload Scheduler for z/OS exits, refer to Chapter 6, “Tivoli Workload Scheduler for z/OS exits” on page 153.

EXTMON(NO)

Currently the only external monitor supported by Tivoli Workload Scheduler for z/OS is Tivoli Business System Manager (TBSM). If you have TBSM installed, specifying YES here will cause Tivoli Workload Scheduler for z/OS to send TBSM information about any alerts or the status changes of jobs that have been flagged to report to an external monitor.

For more information about Tivoli Business System Manager integration, refer to *Integrating IBM Tivoli Workload Scheduler with Tivoli Products*, SG24-6648.

RODMTASK(NO) and RODMPARM(stdrodm)

If your installation uses RODM (resource object data manager), then you may wish to allow the Tivoli Workload Scheduler for z/OS Controller to read the data in this database. This will let its Special Resource Monitor track the real system resource that they have been set up to represent. For the initial implementation, this parameter may be left as NO, and the need for this feature can be investigated when a stable system has been established.

Note: Special Resources are defined in Tivoli Workload Scheduler for z/OS to represent system resources, whose availability, shortage, or mutual exclusivity have an effect on the batch flow. Normally the values associated with these Special Resources must be communicated to Tivoli Workload Scheduler for z/OS by an external program updating the Special Resource Monitor. However, the use of RODM enables the Special Resources Monitor to maintain some Special Resources in real time.

GDGNONST(NO)

This is a Tracker parameter. The dataset triggering function of Tivoli Workload Scheduler for z/OS would normally recognize that the closing data set was a GDG because the GDG flag in the JFCB would be set. However, many programs (including some from IBM) do not set this flag. When using dataset triggering with GDGs, it is advisable to set this parameter to YES. This will stop Tivoli

Workload Scheduler for z/OS from looking for the GDG flag, and assume all data sets having the normal GnnnnVnn suffix to be members of a GDG.

Note: Dataset Triggering is a feature of Tivoli Workload Scheduler. A table of generic and fully qualified data set names is built for the Tracker. When a data set that is matched in the table closes, a Special Resource Event is generated and passed to the Controller to action. This may just set the availability of the Special Resource in the Special Resource Monitor, or it may trigger some additional work into the Tivoli Workload Scheduler for z/OS current plan. Refer to 9.1, “Dataset triggering” on page 204 for more about Dataset Triggering.

GMTOFFSET(0)

Used to bypass GMT synchronization. The value is the number of minutes needed to correct the GMT clock on this system. It is unlikely that you will need to consider this parameter.

RCLPASS(NO)

Only to be considered if you are setting up the Restart and Cleanup function. The value YES will be needed if you have jobs with DISP=(,PASS) in the last step.

VARFAIL(& % ?) and VARPROC(NO)

When a job’s JCL is fetched into Tivoli Workload Scheduler for z/OS or when a job is submitted, Tivoli Workload Scheduler for z/OS resolves Tivoli Workload Scheduler for z/OS variables found within the JCL if the VARSUB parameter has a value of YES or SCAN. JCL variables normally are resolved only in the JCL, but if the JCL contains an instream procedure using parameter VARPROC with a value of YES will allow the use of Tivoli Workload Scheduler for z/OS variables within the procedure. Cataloged procedures are *not* affected by this parameter.

When Tivoli Workload Scheduler for z/OS is unable to resolve a variable because it cannot find a value for it in the user-defined Tivoli Workload Scheduler for z/OS variable tables, and it is not a Tivoli Workload Scheduler for z/OS supplied variable, then the job submission will fail and a Tivoli Workload Scheduler for z/OS error code of OJCV will be assigned. This can be an issue if a lot of the jobs contain variables within them, maybe from an MVS SET instruction or because they contain instream sysin that contains many variables. It is possible to switch variable scanning off and on within the JCL, or define non–Tivoli Workload Scheduler for z/OS variables to Tivoli Workload Scheduler for z/OS with a null value to prevent this, but it can be quite a maintenance overhead, especially when Tivoli Workload Scheduler for z/OS and non-Tivoli Workload Scheduler for z/OS variables appear on the same JCL line. The VARFAIL parameter enables you to tell Tivoli Workload Scheduler for z/OS to ignore non-resolved variables that start with one of the listed variable identifiers (you may code one, two, or all

of &, %, or ?) and are not part of a Tivoli Workload Scheduler for z/OS directive or are a variable that another variable is dependent on.

Initially these parameters may be ignored and reviewed later if JCL variables will be used in the environment.

SPIN(NO)

This parameter enables or disables the use of the JESLOG SPIN function of z/OS. JCC (Job Completion Checker) and the Restart and Cleanup function do not support the use of SPIN and will cause an error message when used. Specifying NO, or leaving to default, will cause Tivoli Workload Scheduler for z/OS to add JESLOG=NOSPIN to the job card of every job it submits.

This parameter is valid only when either the JCC or Restart and Cleanup functions have been enabled (RCLEANUP(YES) or JCCTASK(YES)).

5.2.5 FLOPTS

The FLOPTS statement is valid only for a Controller subtask. It defines the communication method that will be used by the FL (Fetch Log) task. The Fetch Log task fetches joblog output from DataStore when requested by a dialog user or by a Restart and Cleanup process.

Note: This statement is valid only if the OPCOPTS RCLEANUP parameter has a value of YES.

The examples that follow show the parameters built in the CONOP member, depending on the choice of XCF or SNA communication.

Example 5-4 FLOPTS - XCF communication

```
/*********************************************************************/
/* FLOPTS: data store connection                                     */
/*********************************************************************/
FLOPTS   DSTGROUP(OPMCDS)
         CTLMEM(OPMCDSC)
         XCFDEST(XCFOPC1.OPMCDSDS)
```

DSTGROUP(xcfgroupname)

This is the name of the XCF group to be used for Controller - DataStore communication.

Attention: XCFGROUPNAME used for Controller - DataStore communication *must* be different from the XCFGROUPNAME used for Controller - Tracker communication.

CTLMEM(xcfmembername)

This parameter identifies the Controller in the XCF group. Its value should match that in the CTLMEM parameter of the DSTOPTS statement in the DataStore parameters.

XCFDEST(trackerdest.DSTdest,trackerdest.DSTdest)

There can only be a single DataStore active in a JES MAS. Your installation may have several JES MAS, each with its own DataStore. A Tracker task is required on every image. This parameter defines Tracker and DataStore pairs so Tivoli Workload Scheduler for z/OS knows which DataStore to use when looking for output that ran on a workstation defined to a particular Tracker destination.

There should be an entry per Tracker, separated from its DataStore by a period, and separated from the next Tracker and DataStore pair by a comma.

A special Tracker destination of eight asterisks, ********, indicates which DataStore to use when the Tracker destination was left blank in the workstation definition (the Controller submitted the job).

Example 5-5 FLOPTS - SNA Communication

```
/*********************************************************************/
/* FLOPTS: data store connection                                     */
/*********************************************************************/
FLOPTS   CTLLUNAM(OPMCFNLU)
         SNADEST(NCFTRK01.OPMCDSLU)
```

CTLLUNAM(nnnnnn)

This parameter identifies the Controller's luname in the VTAM application when using SNA communication. Its value should match that in the CTLLUNAM parameter of the DSTOPTS statement in the DataStore parameters

SNADEST(nnnnnn.nnnnnn)

See "XCFDEST(trackerdest.DSTdest,trackerdest.DSTdest)" on page 112.

5.2.6 RCLOPTS

The RCLOPTS statement is valid only for a Controller subtask. It defines the options that control the Restart and Cleanup feature.

Note: This statement is valid only if the OPCOPTS RCLEANUP parameter has a value of YES.

This feature may not be required in your installation or it may be something that you plan to implement in the future, but it should not be implemented initially. Consider this feature only after a stable Tivoli Workload Scheduler for z/OS environment has been built and verified. As can be seen from the parameters discussed below, you need a good working knowledge of the batch to define some of the values.

Example 5-6 RCLOPTS

```
/*********************************************************************/
/* RCLOPTS: Restart and clean up options                             */
/*********************************************************************/
RCLOPTS  CLNJOBPX(EQQCL)
         DSTRMM(Y)
         DSTDEST(OPMC)
         DDNOREST(DDNRSO1,DDNRSO2)
         DDNEVER(DDNEXO1,DDNEXO2)
         DDALWAYS(DDALWO1,DDALWO2)
```

CLNJOBPX(nnnnnn)

This parameter identifies the Controllers luname in the VTAM application when using SNA communication. Its value should match that in the CTLLUNAM parameter of the DSTOPTS statement in the DataStore parameters.

DSTRMM(Y)

This parameter defines whether the cleanup process will use the RMM (Removable Media Manager) API for tapes or not.

DSTDEST(nnnn)

Tivoli Workload Scheduler for z/OS inserts output statements into a job's JCL on submission to produce a copy of the output to be retrieved by the DataStore task.

The SYSDEST parameter on the DSTOPTS statement for the DataStore task identifies which output destination DataStore collects. The value on this

parameter must match that value, so that the destination of the copy output is set to that which DataStore will collect.

DDNOREST(DD list)

A list of DD cards that make a step not restartable. This parameter is optional.

DDNEVER(DD list)

A list of DD cards that make a step not re-executable. This parameter is optional.

DDALWAYS(DD list)

A list of DD cards that make a step always re-executable. This parameter is optional.

Other RCLOPTS (not in the sample)

There are several other parameters that cause steps to be or not be restartable, re-executable, or protected, and a few other controls, all of which are optional and will be particular to your installation:

DDPRMEM Points to the member of the PARMLIB data set that contains a list of protected DD names. The list can be reloaded using the command **`F opca,PROT(DD=name)`**. DDPRMEM and DDPROT are mutually exclusive.

DSNPRMEM Points to a member of PARMLIB data set that lists the protected data sets. The list can be reloaded using the command **`F opca,PROT(DSN=name)`**. DSNPRMEM and DSNPROT are mutually exclusive.

DDPROT Defines a list of DD names that are protected.

DSNPROT Defines a list of data set names that are protected.

DSTCLASS Used to direct the duplicated output for DataStore to an output class as well as a destination. When using JCC it is possible that JCC could delete a job's output, which would include the duplicate copy, before DataStore has had a chance to write it away. The DSTCLASS should not be one that is checked by JCC.

STEPRESCHECK Allows manual override of the program logic that prevents certain steps from being non-restartable.

SKIPINCLUDE Prevents errors when INCLUDE statements precede jobcards in the JCL.

5.2.7 ALERTS

The ALERTS statement is valid for both Controller and Tracker started tasks.

Three types of alert can be generated by Tivoli Workload Scheduler: a generic alert that may be picked up by NetView®; messages to the started tasks' own message log (identified by the EQQMLOG DD statement in the started tasks procedure), or to SYSLOG (WTO).

The same alerts can be generated to each of the three destinations, or you may choose to send some alerts to some destinations and some to others.

If you are not using NetView, or will not be processing Tivoli Workload Scheduler for z/OS generic alerts, then the GENALERT parameter may be excluded.

We discuss each type of alert, as the cause is the same regardless of their destination.

Example 5-7 Alerts from CONOP

```
/*********************************************************************/
/* ALERTS:  generating Netview,message log and WTO alerts            */
/*********************************************************************/
ALERTS   GENALERT(DURATION
                  ERROROPER
                  LATEOPER
                  OPCERROR
                  QLIMEXCEED)
         MLOG    (DURATION
                  ERROROPER
                  LATEOPER
                  OPCERROR
                  RESCONT
                  QLIMEXCEED)
         WTO     (DURATION
                  ERROROPER
                  LATEOPER
                  RESCONT
                  OPCERROR
                  QLIMEXCEED)
```

DURATION

A duration alert is issued for any job in started status that has been that way for its planned duration times the limit of feedback value (see "LIMFDBK(100)" on

page 126). This JTOPTS value is always used, even when the job has an operation feedback limit applied.

Tip: Coding a duration time of 99 hours 59 minutes 01 seconds prevents a duration alert from being issued. Use this value for jobs or started tasks that are always active.

ERROROPER

Issued when an operation in the current plan is placed in error status.

LATEOPER

Issued when an operation in ready status becomes late. An operation is considered late when it reaches its latest start time and has not been started, completed, or deleted.

Important: Do not use LATEOPER unless your calculated latest start times bear some resemblance to reality.

RESCONT

Only valid for MLOG and WTO alerts. The alert is issued if a job has been waiting for a resource for the amount of time set by the CONTENTIONTIME parameter of the RESOPTS statement.

OPCERROR

Issued when a Controller-to-Tracker subtask ends unexpectedly.

QLIMEXCEED

Issued when a subtask queue exceeds a threshold value. For all but the event writer, the thresholds are set at 10% intervals starting at 10%, plus 95% and 99%. Tivoli Workload Scheduler for z/OS subtasks can queue a total of 32,000 elements. The size of the event writer queue depends on the size of the ECSA area allocated to it at IPL (defined in IEFSSNxx), and alerts are generated after the area becomes 50% full. If it actually fills, a message will indicate how many messages have been lost.

Example 5-8 Alerts from EQQTRAP

```
/*********************************************************************/
/* ALERTS:  generating Netview,message log and WTO alerts            */
/*********************************************************************/
ALERTS   MLOG    (OPCERROR
                  QLIMEXCEED)
```

```
WTO     (OPCERROR
         QLIMEXCEED)
GENALERT(OPCERROR
         QLIMEXCEED)
```

For more information about IBM Tivoli NetView/390 integration, refer to *Integrating IBM Tivoli Workload Scheduler with Tivoli Products*, SG24-6648.

5.2.8 AUDITS

This statement controls how much data is written to the Tivoli Workload Scheduler for z/OS job tracking logs.

Some data will always be written to these files. This is required to recover the current plan to point of failure if some problem renders the current plan unusable.

ACCESS information at either a READ or UPDATE level may also be written to the file for use in problem determination. An audit program is supplied with Tivoli Workload Scheduler for z/OS that deciphers the data in the records and produces a readable report.

The AMOUNT of data written may be tailored to suit your installation requirements. Either the whole record written back to a database will be recorded (DATA) or just the KEY of that record.

The format of these records can be found in *IBM Tivoli Workload Scheduler for z/OS Diagnosis Guide and Reference Version 8.2*, SC32-1261.

One AUDIT statement can be used to set a global value to be used to determine the ACCESS and AMOUNT values for all databases, or individual statements can be used for each database.

Example 5-9 AUDITS

```
/*********************************************************************/
/* AUDIT: Creating AUDIT information for OPC data                    */
/*********************************************************************/
/* AUDIT    FILE(ALL) ACCESS(READ)   AMOUNT(KEY)                    */
AUDIT    FILE(AD)   ACCESS(UPDATE) AMOUNT(KEY)
AUDIT    FILE(CAL)  ACCESS(READ)   AMOUNT(DATA)
AUDIT    FILE(JS)   ACCESS(READ)   AMOUNT(DATA)
AUDIT    FILE(JV)   ACCESS(READ)   AMOUNT(DATA)
AUDIT    FILE(LT)   ACCESS(UPDATE) AMOUNT(DATA)
AUDIT    FILE(OI)   ACCESS(READ)   AMOUNT(KEY)
AUDIT    FILE(PER)  ACCESS(UPDATE) AMOUNT(KEY)
AUDIT    FILE(RD)   ACCESS(READ)   AMOUNT(KEY)
```

```
AUDIT    FILE(VAR)  ACCESS(READ)   AMOUNT(DATA)
AUDIT    FILE(WS)   ACCESS(READ)   AMOUNT(KEY)
AUDIT    FILE(WSCL) ACCESS(READ)   AMOUNT(KEY)
```

5.2.9 AUTHDEF

The AUTHDEF statement controls how Tivoli Workload Scheduler for z/OS resource security is handled for Tivoli Workload Scheduler.

A resource is a feature or function of Tivoli Workload Scheduler. See Chapter 7, “Tivoli Workload Scheduler for z/OS security” on page 163 for more about resources.

Example 5-10 AUTHDEF statement

```
/*********************************************************************/
/* AUTHDEF: Security checking                                        */
/*********************************************************************/
AUTHDEF  CLASSNAME(IBMOPC)
         LISTLOGGING(ALL)
         TRACE(0)
         SUBRESOURCES(AD.ADNAME
                      AD.ADGDDEF
                      AD.GROUP
                      AD.JOBNAME
                      AD.NAME
                      AD.OWNER
                      CL.CALNAME
                      CP.ADNAME
                      CP.CPGDDEF
                      CP.GROUP
                      CP.JOBNAME
                      CP.NAME
                      CP.OWNER
                      CP.WSNAME
                      CP.ZWSOPER
                      ET.ADNAME
                      ET.ETNAME
                      JS.ADNAME
                      JS.GROUP
                      JS.JOBNAME
                      JS.OWNER
                      JS.WSNAME
                      JV.OWNER
                      JV.TABNAME
```

```
LT.ADNAME
LT.LTGDDEF
LT.OWNER
OI.ADNAME
PR.PERNAME
RD.RDNAME
RL.ADNAME
RL.GROUP
RL.OWNER
RL.WSNAME
RL.WSSTAT
SR.SRNAME
WS.WSNAME)
```

CLASS(IBMOPC)

The class value used defines the name of the security resource that protects the Tivoli Workload Scheduler for z/OS resources. All SAF calls made can be identified by the security product in use as having come from this Tivoli Workload Scheduler for z/OS started task. If you require different security rules for different Tivoli Workload Scheduler for z/OS Controllers, then using a different class value will differentiate the Tivoli Workload Scheduler for z/OS subsystems.

LISTLOGGING(ALL)

If the security definitions specify that access to a subresource should be logged, then this parameter controls how.

FIRST indicates that only the first violation will be logged. When doing a list within Tivoli Workload Scheduler, many violations for the same resource could be caused. ALL specifies that all violations should be logged, and NONE that no violations should be logged.

TRACE(0)

A debug parameter, values of 0, no tracing, 4, partial trace, and 8, full trace, into the EQQMLOG data set.

SUBRESOURCES(.....)

This parameter lists those subresources that you want to protect from unauthorized access.

Subresource protection is at the data level within the databases or plans.

Tip: Initially get Tivoli Workload Scheduler for z/OS up and running without trying to implement a security strategy around subresources. (Start with them commented out.) When all is stable, look at using the minimum number of subresources to protect the data in Tivoli Workload Scheduler.

5.2.10 EXITS

The EXITS statement is valid for both a Controller and a Tracker. However, most of the exits are valid only for one or the other of the tasks.

The exception is exit zero, EQQUX000, which may be called at the start and stop of either the Controller or the Tracker tasks.

The exits themselves are discussed elsewhere in this book.

Example 5-11 Calling exits

```
/*********************************************************************/
/* EXITS: Calling exits                                              */
/*********************************************************************/
EXITS    CALL01(NO)
         CALL02(NO)
         CALL03(NO)
         CALL07(NO)
         CALL09(NO)
```

The valid values for each CALLnn are:

NO Do not load the exit

YES Load the module called EQQUX0nn

Alternatively, you can use the LOADnn parameter. With this parameter, you cause Tivoli Workload Scheduler for z/OS to load a module of the specified name for the exit, or it will default to loading a module called EQQUX0nn. For example, both of these would both load a program module called EQQUX000:

```
CALL00(YES)
LOAD00
```

However, this would load a program module called EXITZERO:

```
LOAD00(EXITZERO)
```

Table 5-2 Exit nn values

Exit	Comment	Task
00	The start/stop exit	Both
01	The Job submit exit	Controller
02	The JCL fetch exit	Controller
03	Application description Feedback exit	Controller
04	Event filtering exit	Tracker
05	JCC SYSOUT archiving exit	Tracker
06	JCC Incident-Record Create exit	Tracker
07	Operation status change exit	Controller
09	Operation initiation exit	Controller
11	Job tracking log write exit	Controller

Tip: Tivoli Workload Scheduler for z/OS will try to load any exit that is valid for the subtask and that has not been explicitly set to CALLnn(NO). To avoid unnecessary load failure messages, code CALLnn(NO) for all exits relevant to the task.

For more about Tivoli Workload Scheduler for z/OS exits, refer to Chapter 6, “Tivoli Workload Scheduler for z/OS exits” on page 153.

5.2.11 INTFOPTS

Controllers’ global settings for handling the Program Interface (PIF). This statement *must* be defined.

Example 5-12 INTFOPTS

```
/*********************************************************************/
/* INTFOPTS: PIF date option                                         */
/*********************************************************************/
INTFOPTS PIFHD(711231)
         PIFCWB(00)
```

PIFHD(711231)

Defines the Tivoli Workload Scheduler for z/OS highdate, used for valid to dates in Tivoli Workload Scheduler for z/OS definitions.

PIFCWB(00)

Tivoli Workload Scheduler for z/OS works with a two-digit year. To circumvent problems with 2000, 1972 was chosen as the Tivoli Workload Scheduler for z/OS base year, so year 00 is actually 1972. This parameter tells Tivoli Workload Scheduler for z/OS which year is represented by 00 in PIF requests.

5.2.12 JTOPTS

The JTOPTS statement is valid only for Controllers or Standby Controllers. It defines how the Tivoli Workload Scheduler for z/OS Controller behaves and how it submits and tracks jobs. Example 5-13 shows an example of this statement.

Example 5-13 JTOPTS statement

```
/*********************************************************************/
/* JTOPTS: How job behaves at workstation and how they are           */
/*         submitted and tracked                                     */
/*********************************************************************/
JTOPTS   BACKUP(1000)
         CURRPLAN(CURRENT)
         DUAL(NO)
         ERRRES(S222,S322)
         ETT(YES)
         HIGHRC(0)
         JOBCHECK(SAME)
         JOBSUBMIT(YES)
         JTLOGS(5)
         LIMFDBK(100)
         MAXJSFILE(NO)
         NEWOILIMIT(30)
         NOERROR(U001,ABC123.*.*.0016,*.P1.S1.U*)
         OFFDELAY(1)
         OUTPUTNODE(FINAL)
         OPINFOSCOPE(IP)
         OPRESTARTDEFAULT(N)
         OPREROUTEDEFAULT(N)
         OVERCOMMIT(0)
         PLANSTART(6)
         PRTCOMPLETE(YES)
         QUEUELEN(10)
         SHUTDOWNPOLICY(100)
         SMOOTHING(50)
         STATMSG(CPLOCK,EVENTS,GENSERV)
         SUBFAILACTION(E)
         SUPPRESSACTION(E)
```

```
SUPPRESSPOLICY(100)
TRACK(JOBOPT,READYFIRST)
WSFAILURE(LEAVE,LEAVE,MANUAL)
WSOFFLINE(LEAVE,LEAVE,IMMED)
```

BACKUP(1000)

The current plan resides in a VSAM file. There are an active version and an inactive version of the file. In the Controllers started task, these are identified by the EQQCP1DS and EQQCP2DS DD statements. Every so often these two files swap and the active file is copied to the inactive file, which then becomes the active file. At the same time the current job tracking (JT) file is closed and written to the JT archive file, and the next JT file in the sequence is activated. The JT files are identified in the Controllers started task procedure by the EQQJTnn and the archive file by the EQQJTARC DD statements.

This is the Tivoli Workload Scheduler for z/OS current plan backup process. How frequently it occurs depends on this parameter.

Using a numeric value indicates that the backup will occur every nnnn events, where nnnn is the numeric value. This increases the frequency of backups during busy periods and decreases it at quiet times.

Note: Regardless of the value of this parameter, Tivoli Workload Scheduler for z/OS swaps (or synchronizes) the current plan whenever Tivoli Workload Scheduler for z/OS is stopped, started, and enters automatic recovery processing, and when the current plan is extended or replanned.

The default value for this parameter is very low, 400, and may cause the CP to be backed up almost continuously during busy periods. As the backup can cause delays to other Tivoli Workload Scheduler for z/OS processes, such as dialog responses, it is advisable to set a value rather than letting it default; 4000 is a reasonable figure to start with.

There is another value that can be used: NO. This stops Tivoli Workload Scheduler for z/OS from doing an automatic backup of the CP. A job can then be scheduled to run at intervals to suit the installation, using the BACKUP command of the EQQEVPGM program. This should be investigated at a later date, after the installations disaster recovery process for Tivoli Workload Scheduler for z/OS has been defined.

Example 5-14 Using program EQQEVPGM in batch to do a CP BACKUP

```
//STEP1 EXEC PGM=EQQEVPGM
//STEPLIB DD DSN=OPC.LOAD.MODULE.LIBRARY,DISP=SHR
```

```
//EQQMLIB DD DSN=OPC.MESSAGE.LIBRARY,DISP=SHR
//EQQMLOG DD SYSOUT=A
//SYSIN DD *
BACKUP RESDS(CP) SUBSYS(opca)
/*
```

CURRPLAN(CURRENT)

This parameter tells Tivoli Workload Scheduler, at startup, to continue using the current plan from the point it was at when Tivoli Workload Scheduler for z/OS was closed.

The other possible value is NEW. This would cause Tivoli Workload Scheduler for z/OS to start using the new current plan (NCP) data set that was created as part of the last CP extend or replan process. It would then start rolling forward the current plan from the data it had written to the tracking logs (JT files) since that NCP was built.

Value NEW would be used only in a recovery situation, where both the active and inactive CP data sets have been lost or corrupted.

DUAL(NO)

Set this parameter to YES if you want to write duplicate copies of the JT files. This can be useful for disaster recovery if you can write them to disks that are physically located elsewhere.

ERRRES(S222,S322)

This parameter lists error codes that, when encountered, cause the job that has them to be reset to a status of READY (that is, automatically set to run again).

It is unlikely that this is desirable, or at least, not initially.

Tip: It is advisable to comment out the sample of this parameter.

ETT(YES)

This parameter switches on the ETT function within the Controller. Event Triggered Tracking can be used to add additional work to the current plan when an event defined in the ETT table occurs. As this is a very useful feature, this parameter should be left as is.

HIGHRC(0)

When a job ends, Tivoli Workload Scheduler for z/OS knows the highest return code for any of its steps. This parameter defines a default value for the highest acceptable return code for all jobs. This can be overridden in the job's definition.

The value used on this parameter should reflect the normal value for the majority of the batch. If it is normal at your installation to perform job condition code checking in "fail" steps within the jobs, then a value of 4095 may be more appropriate than zero.

Note: The default value for this parameter is 4.

JOBCHECK(SAME)

Tivoli Workload Scheduler for z/OS is not a JCL checking tool, but with the value YES on this parameter, it checks the validity of the job card—that is, that it has a job name and is a JOB statement. Using the value NO prevents this. Using SAME takes this one step further, and Tivoli Workload Scheduler for z/OS checks that the job name on the job card is the same as the job Tivoli Workload Scheduler for z/OS is attempting to submit.

Using SAME ensures that Tivoli Workload Scheduler for z/OS will always be able to check the progress (track) the jobs it submits.

JOBSUBMIT(YES)

Switch on job submission at start-up. Job submission covers the submission of scheduled jobs, started tasks, and WTOs. Job submission can be switched on and off when Tivoli Workload Scheduler for z/OS is active via the services dialog panel.

JTLOGS(5)

This parameter tells Tivoli Workload Scheduler for z/OS how many Job Tracking Logs have been defined. These data sets are identified to Tivoli Workload Scheduler for z/OS on the EQQJTxx DD statements in the Controller procedure.

The JT logs are used in sequence. Tivoli Workload Scheduler for z/OS switches to the next JT log when it performs a backup (see "BACKUP(1000)" on page 123). Tivoli Workload Scheduler for z/OS copies the log data to the data set on the EQQJTARC DD statement in the Controller procedure, and marks the copied log as available for reuse.

A minimum of five JT logs is advised to cover situations where Tivoli Workload Scheduler for z/OS does multiple backups in a very short period of time. If Tivoli Workload Scheduler for z/OS does not have a JT log it can write to, it will stop.

LIMFDBK(100)

This parameter is used with the smoothing parameter to tell Tivoli Workload Scheduler for z/OS what rules should be used to maintain job duration times. This default value can be (but is not often) overridden on the job definition.

It also defines the thresholds for duration ALERT messages. Even if the value is overridden in the job it will not affect the calculations for the ALERT message.

Every job defined to Tivoli Workload Scheduler for z/OS must be given a duration time. When the job runs, the estimated, or planned, duration for that job is compared to the actual duration of the job. If the actual time falls within the calculated feedback limit, then the job's duration time will be adjusted for future runs, using the SMOOTHING value.

The LIMFDBK value is any number from 100 to 999, where a value of 200 would set the calculated feedback limit of a 10-minute job to a value between 5 minutes and 20 minutes. The 200 equates to half or double the estimated time. A value of 500 equates to one fifth or 5 times the estimated time. A value of 999 (1000) equates to 1/10 or 10 times the estimate value.

If the actual time for a job falls within the feedback limit, then the difference is applied to the database's estimated time (plus or minus) multiplied by the smoothing factor, which is a percentage value.

Therefore, a LIMFDBK of 500 and a SMOOTHING value of 50 on a 10-minute job that actually took 16 minutes would result in a change to the job's estimated duration, making it13 minutes.

For 16 minutes, you have a limit of 1/5 (2 minutes) and 5 times (50 minutes). The difference between the estimate and the actual was +6 minutes, multiplied by the 50% smoothing value, for a value of +3 minutes to add to the job's duration.

Attention: The value of 100 for LIMFDBK in this sample means no new value will be fed back to the job definitions, so the sample value of 50 for SMOOTHING is meaningless.

Tip: An initial value of 999 for LIMFDBK and 50 for SMOOTHING enable Tivoli Workload Scheduler for z/OS to build fairly accurate duration times, until more useful values for your installation can be set.

MAXJSFILE(NO)

When Tivoli Workload Scheduler for z/OS submits a job, it does it with a copy of the JCL that has been put in the JS file. This is a VSAM data set that holds the copy of the JCL for that run of the job. Identified by the EQQJSnDS statements in

the Controllers started task (where n = 1 or 2) these files, like the current plan files, also switch between active and inactive at specific times.

The MAXJSFILE parameter controls this process.

Specifying NO means no swaps are done automatically, and you should schedule the BACKUP command to perform the backup on a schedule that suits your installation.

Example 5-15 The BACKUP command in BATCH for the JS file

```
//STEP1 EXEC PGM=EQQEVPGM
//STEPLIB DD DSN=OPC.LOAD.MODULE.LIBRARY,DISP=SHR
//EQQMLIB DD DSN=OPC.MESSAGE.LIBRARY,DISP=SHR
//EQQMLOG DD SYSOUT=A
//SYSIN DD *
BACKUP RESDS(JS) SUBSYS(opca)
/*
```

The other values that can be used for this parameter are 0 or a value in kilobytes that means the JS files are swapped based on the size of the JS file.

NEWOILIMIT(30)

Jobs can have documentation associated with them. The use of these operator instructions (OI) is up to the installation. However, if used, when they have been created or changed for a period of time (identified in days by this parameter), then their existence is displayed in lists with a + symbol instead of a Y.

NOERROR(U001,ABC123.*.*.0016,*.P1.S1.U*)

This parameter provides a list of jobs, procedures, steps, and error codes that are not to be considered errors if they match this definition.

For more information, see 5.2.13, “NOERROR” on page 132.

Tip: Comment out this parameter because the sample may not match your installation’s requirements.

OFFDELAY(1)

If a workstation goes OFFLINE, then Tivoli Workload Scheduler for z/OS can reroute its work to an alternate workstation. This parameter says how long Tivoli Workload Scheduler for z/OS will delay action after the change to offline in case the status changes again.

OUTPUTNODE(FINAL)

FINAL or ANY, if Tivoli Workload Scheduler for z/OS processes the first A3P event it receives or if it waits until it receives the one from the FINAL destination of the output.

Restriction: When using Restart and Cleanup, this parameter defaults to FINAL.

A job's output can trigger the JES2 exit7 on every system it passes through on its way to its final destination. When using JCC (Job Completion Checker), using ANY would mean having to have the same JCC definitions for every Tracker. However, if the output is delayed getting to its final destination, then that delay will be reflected in the job's duration time and can delay the successor jobs.

Note: Job Completion Checker is a Tracker function that reads a job's output, scanning for strings of data that may indicate that the job failed or worked, in contrast to the return code of the job.

OPINFOSCOPE(IP)

Operations in the current plan have a user field that may be updated by external process, using the OPINFO command. This parameter determines whether Tivoli Workload Scheduler for z/OS should check jobs in ALL statuses when looking for the operation to update, or if it should restrict its search to only jobs that are currently In Progress (IP).

Note: An IP operation is one in status R A * S I or E.

The OPINFO command is commonly used to pass back data about a problem record (the record number) to Tivoli Workload Scheduler for z/OS when some form of automatic problem raising has been built, probably using Tivoli Workload Scheduler for z/OS exit EQQUX007, the operation status change exit.

For initial installation, you probably do not need be concerned with the value on this parameter.

OPRESTARTDEFAULT(N)

OPREROUTEDEFAULT(N)

These parameters define the default value to be used when defining jobs with regard to their potential for restart and reroute in the case of their normal workstation becoming unavailable through failure or being set offline.

Use N if, in general, jobs should not be eligible for automatic reroute or restart.
Use Y if, in general, jobs should be eligible for automatic reroute or restart.

These value for these parameters are easily overridden in the job's definition.

OVERCOMMIT(0)

Workstations in Tivoli Workload Scheduler for z/OS have a maximum of 99 parallel servers that define how many concurrent operations may be active on that workstation at any one time.

Overcommit affects all workstations with the *automatic reporting* attribute. It adds the overcommit value (0-9999) to the number of defined parallel servers (0-99).

PLANSTART(6)

The time the Daily Planning reports will be calculated from and to. It must match the PLANHOUR parameter on the BATCHOPT statement.

Important: This is not the value used for the planning horizon; it simply defines the reporting timeframe.

PRTCOMPLETE(YES)

This parameter has a meaning only if you are tracking output printing within Tivoli Workload Scheduler. It determines whether the purging of an output group will set the print to complete (YES) or if it may only be set to complete on a print ended event (NO).

QUEUELEN(10)

The current plan (CP) is the interactive and real-time view of the work scheduled to run. Many Tivoli Workload Scheduler for z/OS subtasks want to update the current plan as they receive or process data. Each task has to enqueue on the current plan resource. When they get their turn they do as much work as they are able.

This parameter controls how much work the WSA (Work Station Analyzer) subtask may do. This task is submitting ready jobs to the system. This value says how many jobs it may submit before relinquishing the CP lock. So in this case it will submit 10 jobs, or as many as are ready if that value is less than 10.

Initially this value may remain as is (the actual default is 5); however, when in full production, this parameter may be investigated further.

SHUTDOWNPOLICY(100)

This parameter should definitely be investigated in relation to your installation policy regarding IPLs.

Tivoli Workload Scheduler for z/OS considers how much time is left on a workstation before it closes (parallel servers get set to zero) when determining whether it may submit a job. The parameter value is used as a percentage against the job's duration time. If the calculated value is less than the amount of time left on the workstation, the job will be submitted; if more, it won't.

Workstations have intervals defined against them; where the expectation of an IPL can be pre-scheduled, these two elements together can ensure that batch "runs down" to ensure that none is running when the IPL is scheduled.

SMOOTHING(50)

See discussion in "LIMFDBK(100)" on page 126 with regard to the LIMFDBK parameter.

STATMSG(CPLOCK,EVENTS,GENSERV)

Controls which statistics messages, if any, are generated into the Tivoli Workload Scheduler for z/OS message log.

The STATIM(nn) parameter is used in association with STATMSG to control how frequently the statistics messages are issued.

SUBFAILACTION(E)

If Tivoli Workload Scheduler for z/OS is unable to submit a job (for example, if it cannot find the JCL, or if fails the JOBCHECK check), then you have a choice of how that is reported. The operation in Tivoli Workload Scheduler for z/OS may be put into status E (Error), C (Completed), or left in R (Ready) with the extended status of error. The actual values used depend on whether you use the Tivoli Workload Scheduler for z/OS submit exit, EQQUX001, and if you want its return codes honored or ignored.

The UX001FAILACTION(R) parameter may be used in conjunction with this parameter.

SUPPRESSACTION(E)

On jobs defined to Tivoli Workload Scheduler, one of the settings is "suppress if late." This parameter defines what action is taken for jobs that have this set, and which are indeed late.

The choices are E (set to error), C (Complete), or R (leave in Ready with the extended status of L, Late).

SUPPRESSPOLICY(100)

This parameter provides a percentage to use against the job's duration time, to calculate at what point a job is considered late.

TRACK(JOBOPT,READYFIRST)

The first value may be JOBOPT, OPCASUB, or ALL (the default), which tells Tivoli Workload Scheduler for z/OS whether the jobs are to be tracked.

OPCASUB is used if every job defined in the schedules is submitted by Tivoli Workload Scheduler.

JOBOPT is used when some are and some are not, but you know in advance which these are, and have amended the SUB=Y (default on job definitions) to SUB=N where the job will be submitted by a third party but tracked by Tivoli Workload Scheduler.

Using ALL assumes that all events are tracked against jobs in the plan, regardless of the submitter. In some cases using ALL can cause OSEQ (Out of SEQuence) errors in the plan.

The second parameter has meaning only when using JOBOPT or ALL for the first parameter. It determines how Tivoli Workload Scheduler for z/OS will make the match for events received, with jobs submitted outside Tivoli Workload Scheduler, with jobs in the current plan.

The options are READYFIRST, READYONLY, and ANY.

With READYFIRST and READYONLY, Tivoli Workload Scheduler for z/OS first attempts to find a matching job in R (Ready) status for initialization (A1) events received for jobs submitted outside Tivoli Workload Scheduler. With READYONLY, the search ends there; with READYFIRST, if no match is found Tivoli Workload Scheduler for z/OS goes on to look at jobs in W (Waiting) status. If more than one match is found in R status (or in W status if no R operations), then the operation deemed to be most urgent is selected.

Using ANY, the most urgent operation regardless of status is selected.

WSFAILURE(LEAVE,LEAVE,MANUAL)

WSOFFLINE(LEAVE,LEAVE,IMMED)

These parameters control how jobs that were active on a workstation when it went offline or failed are treated by Tivoli Workload Scheduler.

5.2.13 NOERROR

There are three ways to define NOERROR processing: in the JTOPTS statement, with the NOERROR parameter; with this NOERROR statement; or within members defined to the INCLUDE statements.

Whatever method is chosen to provide Tivoli Workload Scheduler for z/OS with NOERROR information, the result will be the same: a list of jobs, procedure names, and step names, that are not in error when the specified error code occurs.

The method that is chosen will affect the maintenance of these values.

When using the NOERROR parameter on the JTOPTS statement, the noerror list is built when the Controller starts, and changes require the Controller to be stopped and restarted.

When using this statement, the list may be refreshed while the Controller is active, and reloaded using a modify command, **F opca,NEWNOERR** (where opca is the Controller started task name).

When using the INCLUDE statement, NOERROR statements such as this one are built in separate members in the EQQPARM data set concatenation. Each member may be refreshed individually and then reloaded using a modify command **F opca,NOERRMEM(member)**, where opca is the Controller started task and member is the member containing the modified NOERROR statements.

Example 5-16 indicates that an abend S806 in any step, in any proc, in job JOBSUBEX is not to be considered an error.

Example 5-16 NOERROR from CONOP

```
/*********************************************************************/
/* NOERROR: Treating job-tracking error codes as normal              */
/*          completion code                                          */
/*********************************************************************/
/* NOERROR  LIST(JOBSUBEX.*.*.S806)                                  */
```

5.2.14 RESOPTS

RESOPTS statement controls how Special Resources, and operations that use them, are processed (Example 5-17).

Example 5-17 RESOPTS

```
/*********************************************************************/
/* RESOPTS: controlling Special resources                            */
/*********************************************************************/
RESOPTS  ONERROR(FREESR)
         CONTENTIONTIME(15)
         DYNAMICADD(YES)
```

ONERROR(FREESR)

When a job fails that has a Special Resource assigned to it, what should happen to that relationship? The Special Resource may be kept (KEEP) with the job, or freed for use by another job. The free can be unconditional (FREESR), or only when the operation had shared (FREERS) or exclusive (FREERX) use of the Special Resource.

This value is an installation-wide default that may be overridden on the Special Resource definition, or on the job using the Special Resource.

CONTENTIONTIME(15)

Defines how long in minutes Tivoli Workload Scheduler for z/OS will wait before issuing a contention message about a Special Resource preventing a job from being run due to being used by another job.

DYNAMICADD(YES)

If a Special Resource is not defined in the Special Resource database, should Tivoli Workload Scheduler for z/OS add an entry for it in the current plan's Special Resources Monitor when a job has it as a requirement or if a Special Resource event is received.

The other options are NO, EVENT, and OPER. OPER says only create the entry if it is an operation that wants to use the resource. EVENT says only create it if it is in response to a Special Resource event.

5.2.15 ROUTOPTS

In the Controller, the ROUTOPTS statement (Example 5-18) tells it who may communicate with it and what method they will use to communicate.

If the communication is via SNA, then in the OPCOPTS statement, the NCF task will have been started, and the Controllers LU name will have been provided using the NCFTASK and NCFAPPL parameters.

If the communication is via XCF, there is no task to start; however, the XCFOPTS statement will be required to state the name of the Controller/Trackers' XCF and the XCF group.

Note: The Controller does not initiate communication. Other tasks, such as the Tracker, look for the Controller, and start the conversation.

Example 5-18 ROUTOPTS

```
/*********************************************************************/
/* ROUTOPTS: Communication routes to tracker destinations            */
/*********************************************************************/
   ROUTOPTS SNA(NCFTRK01)
/*-------------------------------------------------------------------*/
/* ROUTOPTS APPC(AS4001:MYNET.MYLU)                                  */
/*          CODEPAGE(IBM-037)                                        */
/*          DASD(EQQSUES5,EQQSUES6)                                  */
/*          OPCAV1R2(NECXMTB1,EQQSUR2)                               */
/*          PULSE(5)                                                 */
/*          SNA(NCFXMTA3,NCFXMTB1,NCFXMTB3)                          */
/*          TCP(AIX1:99.99.99.99)                                    */
/*          TCPID(TCPPROC)                                           */
/*          TCPIPPORT(424)                                           */
/*          USER(BIKE42,VACUUM)                                      */
/*          XCF(TRK1,TRK2)                                           */
/*-------------------------------------------------------------------*/
/* The way the SYSCLONE variable can be used to identify in a shared */
/* parm environment the different XCF trackers is for example, if    */
/* the trackers are 2 and their SYSCLONE value is K1 and K2, to      */
/* specify in ROUTOPTS:                                              */
/*                                                                   */
/* XCF(TRC1,TRC2)                                                    */
/*                                                                   */
/* and in Tracker XCFOPTS  (see tracker EQQTRAP sample):             */
/*                                                                   */
/* MEMBER(TR&SYSCLONE)                                               */
```

```
/*-------------------------------------------------------------------*/
/*********************************************************************/
/* XCFOPTS: XCF communications                                       */
/*********************************************************************/
/* XCFOPTS  GROUP(OPCESA)                                            */
/*          MEMBER(XCFOPC1)                                          */
/*          TAKEOVER(SYSFAIL,HOSTFAIL)                               */
```

All of the following parameters list the identification for the Trackers that may connect to them and that are used to connect a destination and workstation definition.

DASD(subrel1,subrel2,....)

A DASD identifier is actually a DD statement in the Controllers procedure. The Controller writes JCL and synchronization activities to a sequential data set, and the appropriate Tracker reads from that data set. On the Controller end of the communication, the DD name, the identifier, can be any name that the installation finds useful to use. There may be up to 16 Trackers connected to the Controller in this way. On the Tracker end of the communication, the DD name will be EQQSUDS.

SNA(luname1, luname2,....)

The VTAM LU names defined for Trackers that may connect to the Controller.

USER(user1,user2,....)

This parameter identifies the different destinations that have been built making use of the Tivoli Workload Scheduler for z/OS open interface. The Controller will use the operation initiation exit, EQQUX009, to communicate with this destination.

XCF(member1,member2,....)

The XCF member names defined in the Trackers XCFOPTS statement on the MEMBER parameter.

PULSE(10)

How frequently, in minutes, the Controller expects to get a handshake event from the Trackers. Missing two consecutive handshakes causes the Controller to force the destination offline. This can initiate reroute processing in the Controller.

5.2.16 XCFOPTS

In the EQQCONOP sample this statement and parameters are commented out, but as XCF communication is normally the quickest and easiest to set up, it is quite likely you will wish to change that.

When using XCF for communication, you must code the XCFOPTS statement in the Controller and in the Trackers that are communicating with it by this method.

The XCFOPTS statement in the EQQTRAP sample is exactly the same as the one in EQQCONOP except for the value in the MEMBER parameter.

GROUP(OPCESA)

The name you have decided to use to identify your Controller and Tracker communication. This value will be the same for both the Controller and the Tracker.

Note: XCF communication can be used for Controller - Tracker communication, and for Controller - DataStore communication. These are completely separate processes. A different XCF Group is needed for each.

MEMBER(nnnnnn)

A name to uniquely identify each member of the group.

When used for a Tracker, this is how the Tracker is identified in other statements, such as the XCF parameter in the ROUTOPTS statement above. Here you would list the member names of the Trackers allowed to connect to this Controller via XCF.

All scheduled activities in Tivoli Workload Scheduler for z/OS take place on a workstation. You may have a workstation to represent each system in your environment. The relationship between the Tracker and the workstation is made by using the Tracker's member name in the workstation's destination field. The Controller then knows that JCL for jobs scheduled on that workstation is to be passed across that communication pathway.

TAKEOVER(SYSFAIL,HOSTFAIL)

This parameter is valid for Standby Controllers. It automates the takeover by this Standby Controller when it detects that the Controller has failed or that the system that hosts it has failed.

In these situations, if this parameter were not coded, the Standby Controller would issue a system message so that the takeover could be initiated manually.

5.3 EQQCONOP - STDAR

This member in EQQCONOP (Example 5-19) contains the statement that controls the Auto Recovery feature of Tivoli Workload Scheduler. Activating this feature with RECOVERY(YES) in OPCOPTS, does not cause jobs to start recovering automatically from failures.

Jobs using this feature have Tivoli Workload Scheduler for z/OS recovery directives coded in their JCL. These directives identify specific failures and associate recovery activities with them.

These parameters should be placed in a member called STDAR, or in a member identified in OPCOPTS by the ARPARM(member) parameter.

Example 5-19 AROPTS member of the EQQCONOP sample

```
/*********************************************************************/
/* AROPTS: Automatic job recovery options                            */
/*********************************************************************/
AROPTS AUTHUSER(JCLUSER)
       ENDTIME(2359)
       EXCLUDECC(NOAR)
       EXCLUDERC(6)
       PREDWS(CPU*)
       STARTTIME(0000)
       USERREQ(NO)
```

AUTHUSER(JCLUSER)

What value should Automatic Recovery use to check for the authority to perform its recovery actions? There are four possible values:

JCLUSER The name of the user who created or last updated the JCL in Tivoli Workload Scheduler. If no one has updated the JCL, then Tivoli Workload Scheduler for z/OS uses the user in the PDS copy of the JCL. If there are no ISPF statistics, then Tivoli Workload Scheduler for z/OS does not perform authority checking.

JCLEDITOR The name of the user who created or last updated the JCL in Tivoli Workload Scheduler. If no one has updated the JCL then Tivoli Workload Scheduler for z/OS does not perform authority checking.

OWNER The first 1-8 characters of the owner ID field of the application.

GROUP The value in the authority group ID field.

ENDTIME(2359)

This parameter along with the STARTTIME parameter control when auto recovery may take place for jobs where the TIME parameter has not been included on the RECOVERY directive.

A STARTTIME of 0000 and an ENDTIME of 2359 mean that auto recovery is effective at all times.

EXCLUDECC(NOAR)

RECOVERY directives in the JCL can be specific about the error codes that will be recovered, or they can be matched generically. It is often the case that a job may have specific and generic RECOVERY directives.

Using the EXCLUDECC enables you to specify a code or a group of codes defined in a *case code* that will be excluded from automatic recovery unless specified explicitly on the RECOVERY directive.

That is, this code or group of codes will never match a generic directive.

The supplied default case code NOAR, contains the following codes: S122, S222, CAN, JCLI, JCL, and JCCE. Additional codes may be added to the NOAR case code. See *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265 for more information about case codes.

EXCLUDERC(6)

This parameter defines the highest return code for which no auto-recovery will be done unless specifically coded on the RECOVERY directive. In this case condition codes 1-6 will trigger automatic recovery actions only if they were coded explicitly on the RECOVER directive, as condition codes 1-6 will not match any generic entries.

PREDWS(CPU*)

Automatic Recovery can add an application to the current plan in response to a particular failure situation. It is expected that this application contains jobs that should be processed before the failing job is rerun or restarted.

If the added application does not contain a job that is defined as a predecessor to the failed operation, then Tivoli Workload Scheduler for z/OS uses this parameter value to determine which of the operations in the application should be the failed job's predecessor.

The job with the highest operation number using a workstation that matches the value of this parameter is used. The value, as in the sample, may be generic, so

the operation with the highest number on a workstation whose name starts CPU will be used.

If no jobs match the workstation name, then the operation seen as the last operation (one with no internal successors) is used, or if there are multiple “last” operations, the one with the highest operation number is used.

STARTTIME(0000)

See discussion in “ENDTIME(2359)” on page 138.

USERREQ(NO)

Must Tivoli Workload Scheduler for z/OS have determined a user ID to perform authority checking against when it needs to update the current plan? See “AUTHUSER(JCLUSER)” on page 137 for more information.

CHKRESTART(NO)

This is the only AROPTS parameter missing from the sample. It has meaning only if the Restart and Cleanup function is in use.

Use NO to always POSTPONE the recovery actions specified when cleanup activities are required.

Use YES to POSTPONE the recovery actions only when a job restart is needed.

5.4 EQQCONOP - CONOB

These parameters are not used by either the Controller or the Tracker. They are used by the batch jobs that build the Tivoli Workload Scheduler for z/OS long-term and current plans, plus other Tivoli Workload Scheduler for z/OS batch processes.

When running EQQJOBS, option two, you built the skeleton files that will be used when dialog users want to submit Tivoli Workload Scheduler for z/OS batch jobs. During this process you decided the name to be used for this parameter member, which will have been inserted into the built JCL. This CONOB section of EQQCONOP should be copied into a member of that name.

Example 5-20 BATCHOPT

```
/*********************************************************************/
/* BATCHOPT: Batch jobs options                                      */
/*                                                                   */
/* See EQQE2EP sample for the TPLGY member and for other parameters  */
/* connected to the END-TO-END feature.                              */
```

```
/*********************************************************************/
BATCHOPT CALENDAR(DEFAULT)
         CHECKSUBSYS(YES)
         DATEFORM('YY/MM/DD')
         DPALG(2)
         DPROUT(SYSPRINT)
         DYNAMICADD(YES)
         HDRS('LISTINGS FROM SAMPLE',
              'OPC',
              'SUBSYSTEM CON')
         LOGID(01)
         LTPDEPRES(YES)
         NCPTROUT(YES)
         OCPTROUT(CMP)
         OPERDALL(Y)
         OPERIALL(Y)
         PAGESIZE(55)
         PLANHOUR(6)
         PREDWS(CPU*)
         PREVRES(YES)
         SUBSYS(CON)
         SUCCWS(CPU*)
         VALEACTION(ABEND)
         TPLGYPRM(TPLGY)
/*********************************************************************/
/* RESOURCE: Controlling Special Resources                           */
/*********************************************************************/
RESOURCE FILTER(TAPE*)
```

CALENDAR(DEFAULT)

When building jobs into applications in TWS, you associate a calendar to the application. A calendar determines the working and non-working (free) days of the business. If the calendar field is left blank, then the value in this parameter is used to determine the working and non working (free) days applied to that application when the long-term plan is built.

The TWS planning jobs will consider all days to be working days if no default calendar has been built in the calendar database and named in this parameter.

CHECKSUBSYS(YES)

If the current plan extension job runs on a system that cannot communicate properly with the Controller subsystem, then the current plan may be corrupted.

Using YES for this parameter (the default is NO) will prevent the batch job from amending the files when it cannot communicate with the Controller.

DATEFORM('YY/MM/DD')

The default date format used in reports created by TWS planning and reporting jobs is YY/MM/DD. This parameter enables you to amend this to the format common in your installation.

DPALG(2)

When it builds its current plan, TWS calculates the relative priority of every job in its schedule based on the duration time of each job and the deadline time of the final jobs in the networks. However, each application has a priority value defined against it. This parameter balances how much this value influences the way the plan is built. The sample (and default) value of 2 provides a reasonable balance.

DPROUT(SYSPRINT)

This parameter identifies the DD name used for the reports created by the current plan extend batch.

DYNAMICADD(YES)

When the current plan is built, any Special Resources used by jobs in the plan are added to the Special Resource Monitor. Those with a definition in the Special Resource database are added according to that definition. Those with no definition, dynamic special resources, are added to the monitor with default values. Alternatively the dynamic special resources can be added to the monitor in real time when the first job that uses it, runs.

Where an installation uses a very large number of special resources, then this process can add considerably to the time taken to create the new current plan. In this case, consider using NO for this parameter.

HDRS('.....')

Up to three lines that the user can use to personalize the reports from the current plan batch jobs.

LOGID(01)

Every activity in TWS may be audited (see 5.2.8, "AUDITS" on page 117). They are written to the JTlogs during real-time ("JTLOGS(5)" on page 125) and when the current plan switches, to the JTARC file ("BACKUP(1000)" on page 123). When the current plan is extended, these logs are no longer needed for forward recovery of the plan and will be lost, unless written to a further log, the EQQTROUT DD card in the planning job, that may be used for auditing.

When there are several TWS Controllers in an installation, and all of them collect their data into the same EQQTROUT file, then using different values for this parameter identifies which Controller each record came from.

LTPDEPRES(YES)

The long-term plan in TWS can be extended and it can be modified. The most up-to-date long-term plan should be fed into the current plan extend process. A modify may be run any time changes are made to the databases to ensure that these are included, but it is more normal to run it just before the current plan is extended.

The long-term plan must also be extended regularly; it must never run out. Using YES as the value to this parameter causes a modify of the existing long-term plan as well as extend, when an extend is run.

Tip: Using YES for this parameter and extending the long-term plan by one day, daily, before the current plan extend, removes the need to run a modify every day.

NCPTROUT(YES)

Should some records regarding the new current plan be written to the EQQTROUT file when the current plan is extended?

OCPTROUT(CMP)

Should some records be copied to the EQQTROUT file when the current plan is extended?

OPERDALL(Y)

Where a job deadline has been set for tomorrow, TWS needs to know whether tomorrow means tomorrow, or if it means the next working day according to the calendar in use. The value of N will cause the +x days deadline for a job to be moved so that it skips non-working days. With a value of Y, tomorrow really means tomorrow.

OPERIALL(Y)

The same calculation as for OPERDALL, but for an operation whose input arrival time has been defined as “plus x days.”

PAGESIZE(55)

Number of lines in a page of a TWS report.

PLANHOUR(6)

Defines the time TWS uses as a cutoff when building the reports from previous planning periods. This value must match the one in the PLANSTART parameter in JTOPTS.

PREDWS(CPU*)

Where a defined predecessor cannot be found, this parameter defines which workstation should be used (the value may be generic) to find a substitute. The last operation on this workstation in the predecessor application will be used instead to build the predecessor dependency.

PREVRES(YES)

Whether the reports produced cover the whole of the previous 24 hours.

SUBSYS(CON)

Identifies which TWS controller subsystem the batch job using this parameter member is to process against. The controller must exist on the same system or in the same GRS ring.

SUCCWS(CPU*)

The name of the workstation to use; may be generic when a defined successor cannot be found. The first operation found on the workstation in the successor application will be used as a substitute.

VALEACTION(ABEND)

The action the planning batch job should take if its validation code detects an error in the new plan.

TPLGYPRM(TPLGY)

When using the TWS End-to-End feature, this parameter names the member where the topology statements and definitions may be found.

5.5 RESOURCE - EQQCONOP, CONOB

FILTER(TAPE*)

The resource statement lists the resources that reports are required for. The resource name may be generic.

5.6 EQQTRAP - TRAP

Example 5-21 shows the full TRAP section of the EQQTRAP sample member.

The OPCOPTS, ALERTS, and EXITS statements have been discussed previously, and only the Tracker end of the communication statements and parameters remains to be discussed from this sample.

Example 5-21 The TRAP member of the EQQTRAP sample

```
/*********************************************************************/
/* OPCOPTS: run-time options for the TRACKER processor               */
/*********************************************************************/
OPCOPTS  OPCHOST(NO)
         ERDRTASK(0)
         EWTRTASK(YES)      EWTRPARM(STDEWTR)
         JCCTASK(YES)       JCCPARM(STDJCC)
          NCFTASK(YES)       NCFAPPL(NCFTRK01)
/*-------------------------------------------------------------------*/
/* If you want to use Automatic Restart manager you must specify:    */
/*       ARM(YES)                                                    */
/*-------------------------------------------------------------------*/
/*********************************************************************/
/* TRROPTS: Routing option for communication with Controller         */
/*********************************************************************/
TRROPTS  HOSTCON(SNA)
         SNAHOST(NCFCNT01)
/*-------------------------------------------------------------------*/
/* If you want to use DASD connection you must specify:              */
/*       HOSTCON(DASD)                                               */
/*-------------------------------------------------------------------*/
/* If you want to use XCF connection you must specify:               */
/*       HOSTCON(XCF)                                                */
/*                                                                   */
/* and add the XCFOPTS statement too                                 */
/*-------------------------------------------------------------------*/
/*********************************************************************/
/* XCFOPTS: XCF communications                                       */
/*********************************************************************/
/*-------------------------------------------------------------------*/
/* XCFOPTS GROUP(OPCESA)                                             */
/*          MEMBER(XCFOPC2)                                          */
/*          TAKEOVER(SYSFAIL,HOSTFAIL)                               */
/*-------------------------------------------------------------------*/
/*********************************************************************/
```

```
/* ALERTS:  generating Netview,message log and WTO alerts           */
/*******************************************************************/
ALERTS   MLOG    (OPCERROR
                  QLIMEXCEED)
         WTO     (OPCERROR
                  QLIMEXCEED)
         GENALERT(OPCERROR
                  QLIMEXCEED)
/*******************************************************************/
/* EXITS: Calling exits                                            */
/*******************************************************************/
EXITS    CALL00(NO)
         CALL04(NO)
         CALL05(NO)
         CALL06(NO)
```

5.6.1 TRROPTS

This parameter defines how the Tracker communicates with the Controller.

HOSTCON(comms type)

Choose, SNA, XCF, or DASD.

DASD does not need any further parameters, The Tracker will know to write to the EQQEVDS data set, and read from the EQQQSUDS data set.

XCF requires the coding of the XCFOPTS statement.

SNA requires the SNAHOST parameter to identify the luname of the Controller (plus standby Controllers if used).

The OPCOPTS statement NCFTASK(YES) and NCFAPPL(luname) are needed for SNA communication. This luname identifies this Tracker. It should match an entry in the Controllers ROUTOPTS statement.

Note: If using DataStore, then this luname name will appear, paired with a DataStore destination, in the FLOPTS statement on the SNADEST parameter.

5.6.2 XCFOPTS

XCFOPTS identifies the XCF group and member name for the Tivoli Workload Scheduler for z/OS started task.

GROUP(group)

The XCF group name identifying Controller / Tracker communication.

MEMBER(member)

The XCF member name of this Tracker. It should match an entry in the Controllers ROUTOPTS statement.

Note: When using DataStore, this member name will appear, paired with a DataStore destination, in the FLOPTS statement on the XCFDEST parameter.

TAKEOVER

Not valid for a Tracker task.

5.7 EQQTRAP - STDEWTR

This member of EQQTRAP defines the Event Writer Task for the Tracker.

Some parameters are valid only for the EWTROPTS statement when the submit data set is being used. This data set is needed only when using shared DASD communication between the Controller and Tracker.

Important: In the notes below some event types are mentioned: A4, A3S, A3P, and so forth. These are the event names for a JES2 installation. If you are using JES3, then the event names will be B4, B3S, B3P, and so on.

Example 5-22 EWTROPS from EQQTRAP sample

```
/*********************************************************************/
/* EWTROPTS: Event Writer task options                               */
/*********************************************************************/
EWTROPTS EWSEQNO(01)
         HOLDJOB(USER)
         RETCODE(HIGHEST)
         STEPEVENTS(NZERO)
```

EWSEQNO(01)

Using this parameter tells the Tracker to start an Event Writer with the Event Reader function.

The Tracker will write events to the event data set and pass them to the Controller immediately, via the SNA or XCF connection in place.

The event data set is identified in the Trackers procedure by DD statement EQQEVDS.

If communication between the Tracker and Controller fails, or the Controller is unavailable for some reason, then the Tracker will continue to write events to the event data set. After communication has been re-established the Tracker will check the last read/last written flags in the event records and resynchronize with the Controller.

Attention: The event data set is a wraparound file. The first time the Tracker starts, it will format this file and an E37 will be received. This is normal.

HOLDJOB(USER)

The default value for this parameter is NO. This stops Tivoli Workload Scheduler for z/OS from holding and releasing jobs not submitted by Tivoli Workload Scheduler. However, normally not every job that you want to include and control in your batch schedule is submitted by Tivoli Workload Scheduler.

This parameter has two other values, YES and USER.

Using YES may upset a few people. It causes every job that enters JES to be HELD automatically. Tivoli Workload Scheduler for z/OS then checks to see whether that job should be under its control. That is, a job of that name has been scheduled in Tivoli Workload Scheduler for z/OS and is part of the current plan or defined in the Event Trigger Table. Jobs that should be controlled by Tivoli Workload Scheduler for z/OS are left in hold until all of their predecessors are complete. If the job is not to be controlled by Tivoli Workload Scheduler, then the Controller can tell the Tracker to release it.

Note: The Tracker Holds and Releases the jobs, but it needs information from the Controller because it has no access to the current plan.

Using USER means that Tivoli Workload Scheduler for z/OS does not HOLD jobs that enter JES. Using USER tells the Tracker that any job that will be submitted outside Tivoli Workload Scheduler for z/OS and that will be controlled by Tivoli Workload Scheduler for z/OS will have TYPRUN=HOLD on the job card. When the job is defined in Tivoli Workload Scheduler, it will be specifically flagged as

SUB=NO. The Controller will advise that the job should be released when all of its predecessors are complete.

RETCODE(HIGHEST)

When the event writer writes the A3J event (Job ended event from SMF exit IEFACTRT), it can pass the return code from LAST step processed, of the HIGHEST return code encountered during the job's processing.

This return code will be used by Tivoli Workload Scheduler for z/OS to judge the successful completion of the job (or otherwise).

The best value to use depends on any JCL standards in place in your installation.

When you define a job to Tivoli Workload Scheduler, you state the highest return code value that may be considered acceptable for the job. The return code checked against that value is determined by this parameter.

STEPEVENTS(NZERO)

The other options for this parameter are ALL and ABEND. When job steps complete, Tivoli Workload Scheduler for z/OS may send an A3S event to the Controller. When you use NZERO or ABEND the event is created only for steps that ABEND, or that end with other than a non-zero completion code.

ALL causes an A3S event to be written for every step end. This is needed only if you use Automatic Recovery to detec*t flushed steps*.

PRINTEVENTS(END)

This parameter controls the generation of A4 events. A4 events are created when an output group has printed.

NO No print events will be generated. Use this value if you do not want to track printing.

ALL Print events will be generated and Tivoli Workload Scheduler for z/OS will reflect the time it took to actually print the output group, excluding any time that the printer was interrupted.

END Print events are generated and reflect the time from start to finish of the printing, including time when it was interrupted.

Tip: As it is quite rare to track printing, consider setting this value to NO.

SUREL, EWWAIT, SKIPDATE, and SKIPTIME

These parameters are used only when communication between the Controller and Tracker is done via shared DASD.

The SUREL parameter will have a value of YES, default value is NO.

EWWAIT has a default value of 10 (seconds). This determines how long after the event writer reads the last record in the submit/release data set it will check it again. The submit/release data set is written to by the Controller when it wants the Tracker to submit a job whose JCL it has written there, or release a job that is currently on the HOLD queue. If you are using shared DASD, then reduce this value from 10 as 10 seconds is a long time to wait between jobs these days.

Important: Remember, shared DASD communication is very slow, especially when compared to XCF and SNA, neither of which are complicated or time consuming to set up.

SKIPDATE and SKIPTIME define the limit for old records in the submit/release data set. The Tracker will not submit jobs written before the SKIPTIME on the SKIPDATE.

5.8 EQQTRAP - STDJCC

For initial set-up of Tivoli Workload Scheduler for z/OS it is very unlikely that you will need to use the JCC task. Its need can be evaluated when a basic Tivoli Workload Scheduler for z/OS system is stable.

JCC (Job Completion Checker) is a function of the Tracker tasks. It is a post-processing feature that scans a job's output looking for strings of data that may indicate that a job has worked, or failed, in contradiction to the condition codes of the job.

Example 5-23 JCCOPTS

```
/*********************************************************************/
/* JCCOPTS: Job Completion Checker options:                          */
/*********************************************************************/
JCCOPTS CHKCLASS(A)
        INCDSN(OPC.SAMPLE.INCIDENT)
        JCCQMAX(112)
        JCWAIT(4)
        MAXDELAY(300)
        SYSOUTDISP(HQ)
        UMAXLINE(50)
        USYSOUT(JOB)
```

CHKCLASS(A)
Identifies which classes (up to 16) are eligible to be processed.

INCDSN(data.set.name)
This identifies where a record of the incident may be written. This sequential data set must pre-exist, and may be written to by several different Tivoli Workload Scheduler for z/OS subsystems. The file is not allocated by Tivoli Workload Scheduler, so it may be renamed, deleted, or redefined while Tivoli Workload Scheduler for z/OS is active.

JCCQMAX(112)
JCC intercepts the A3P event before it is written to the event data set and passed to the Controller. This is so it can amend the data in the record before it is processed by the Controller. This value indicates the maximum number of A3P events that may be queued to JCC; 112 is the maximum allowed, and any other value used should be a multiple of 16.

JCWAIT(4)
This parameter is not defined in the sample. It says how long, in seconds, JCC will wait before checking the JES to see if a job's output is available. Four seconds is the default.

MAXDELAY(300)
JCC will check for this number of seconds for output for jobs that it believes should have sysout to check. If the time is reached and no output is forthcoming, then Tivoli Workload Scheduler for z/OS will place the job in error status.

SYSOUTDISP(HQ)
When the output has been processed, what should happen to it—deleted, released, or requeued.

UMAXLINE(50)
Defines how many lines of user output will be scanned, from zero up to 2147328000 lines.

Tip: Try not to write dumps to a sysout class that JCC is checking.

USYSOUT(JOB)
Is user sysout scanned: ALWAYS, NEVER, or only when there is a specific JOB table defined.

6

Tivoli Workload Scheduler for z/OS exits

In this chapter we provide a brief description of each of the exit points within Tivoli Workload Scheduler for z/OS and associated functions.

You will find the following sections in this chapter:

- EQQaaaaa exits
- User-defined exits

6.1 EQQUX0nn exits

EQQUX0nn exits (where nn is a number) are loaded by the Controller or the Tracker subtasks when Tivoli Workload Scheduler for z/OS is started. These exits are called when a particular activity within the Controller or Tracker occurs, such as when an event is received or an operations status changes.

Some exits are valid only for the Controller started task, and some only for the Tracker. Tivoli Workload Scheduler for z/OS loads an exit when requested to do so by its initialization parameters. See 5.2.10, “EXITS” on page 120 for a full description of this statement.

Example 6-1 EXITS parameter

```
EXITS    CALL00(YES)
         CALL01(NO)
         CALL02(YES)
         CALL03(NO)
         LOAD07(EQQUX007)
         CALL09(NO)
```

Table 6-1 lists the EQQUX0nn exits, the subtask that uses them, and their common name.

Table 6-1 EQQUX0nn exits

Exit	Common Name	Task
EQQUX000	The start/stop exit	Both
EQQUX001	The Job submit exit	Controller
EQQUX002	The JCL fetch exit	Controller
EQQUX003	Application Description Feedback exit	Controller
EQQUX004	Event filtering exit	Tracker
EQQUX005	JCC SYSOUT Archiving exit	Tracker
EQQUX006	JCC Incident-Record Create exit	Tracker
EQQUX007	Operation Status Change exit	Controller
EQQUX009	Operation Initiation exit	Controller
EQQUX011	Job Tracking Log Write exit	Controller

6.1.1 EQQUX000 - the start/stop exit

This exit is called when the Controller or Tracker task initiates and when it closes normally. It can be used to set up the environmental requirements of other exits. For example, it can be used to open and close data sets to save repetitive processing by other exits.

The sample EQQUX000 supplied with Tivoli Workload Scheduler, EQQUX0N, is a PL1 program to set a user-defined workstation to active.

A sample EQQUX000 is available with the redbook *Customizing IBM Tivoli Workload Scheduler for z/OS V8.2 to Improve Performance*, SG24-6352, that is used to open data sets for the EQQUX002 sample. This EQQUX002 sample is also available with the redbook. See 6.1.3, “EQQUX002 - the JCL fetch exit” on page 155 for more information on that sample.

6.1.2 EQQUX001 - the job submit exit

When Tivoli Workload Scheduler for z/OS is about to submit a job or start a started task, it will call this exit, if loaded, to alter the JCL being submitted. By default it cannot add any lines to the JCL, but if the EXIT01SZ parameter has been defined on the OPCOPTS statement, then the exit can increase the number of lines, up to the maximum specified on the parameter.

The sample provided with Tivoli Workload Scheduler for z/OS provides methods to:

1. Provide an RUSER value for the job to use rather than having it default to the Tivoli Workload Scheduler for z/OS value. The RUSER may be found from a job name check, using the Authority Group defined for the jobs application, or from the USER=value on the job card.
2. Change the job’s MSGCLASS.
3. Insert a step after the job card, and before the first EXEC statement in the JCL.

6.1.3 EQQUX002 - the JCL fetch exit

The normal process for fetching JCL uses a concatenation of JCL libraries against a single DD called EQQJBLIB. This process may not suit all installations, for various reasons.

Very large installations may find that this I/O search slows the submission as each library index has to be searched to find the JCL. Using this exit to directly access the library required can shorten this process.

Each department may only have security access to one library, and may even duplicate job names within those libraries. Using the exit can prevent the JCL from being selected from the wrong library higher in the concatenation, and amend the JCL to reflect the correct authority.

The library access method for some JCL libraries may be different to the others, needing to be called via a special program. Again the exit may be the solution to this.

JCL may be held in a PDS member, or a model member, whose name differs from the job name to be submitted. The normal JCL fetch expects the member name and the job or operation name to be the same. The exit would be needed to cover this situation too.

The EQQUX002 sample provided with Tivoli Workload Scheduler for z/OS searches another DD statement that you insert in to the Controller started task, called MYJOBLIB, before searching the EQQJBLIB concatenation.

See *Customizing IBM Tivoli Workload Scheduler for z/OS V8.2 to Improve Performance*, SG24-6352, for more information on this exit and a multifunction sample that can deal with many of the situations mentioned above.

6.1.4 EQQUX003 - the application description feedback exit

Tivoli Workload Scheduler for z/OS uses each job's duration time when it calculates the running time of the current plan. The duration time is maintained by Tivoli Workload Scheduler for z/OS using the Limit of Feedback and Smoothing parameters on the JTOPTS statement.

If you require a different algorithm to be used, you may use this exit to amend the value calculated by Tivoli Workload Scheduler for z/OS before the Application Description is updated with the new value.

6.1.5 EQQUX004 - the event filter exit

The Tracker has no knowledge of the current plan, which lives in the Controller. When it takes the SMF, JES, and user events from its ECSA area, it passes them straight onto the Controller without alteration.

This exit enables you to filter the events so that fewer are passed to the Controller. For example, on a testing system, there may only be a few housekeeping jobs whose events are relevant to the schedule.

The sample EQQUX004 supplied with Tivoli Workload Scheduler for z/OS searches for specific job names in the events and only passes those that are matched.

6.1.6 EQQUX005 - the JCC SYSOUT archiving exit

The JCC function of Tivoli Workload Scheduler for z/OS can read the output of a job and determine whether it worked, agreeing or disagreeing with the condition codes within the job. It then acts on its parameters and releases or requeues the output.

This exit allows you to alter this normal behavior and take different actions.

The sample exit EQQX5ASM requeues the job's output to different classes based on the job's success or failure.

6.1.7 EQQUX006 - the JCC incident-create exit

This exit alters the format of the error record written to the JCC incident data set, which will be written if the job definition in the EQQJCLIB data sets indicates that a record should be raised.

The sample members for this exit, EQQX6ASM and EQQX6JOB, create a two-line record for the incident log.

6.1.8 EQQUX007 - the operation status change exit

This Controller exit is called every time an operation in the current plan changes status. It can be used to do a multitude of things.

The most common usage is to pick up when a job changes to ERROR status and automatically generate a problem record.

The sample provided, EQQX7ASM and EQQX7JOB, submits a job to the internal reader when the ERROR status occurs. This job has a dummy SYSIN value of AAAA, which is substituted with the parameter string passed to the exit.

The actions taken by the job result in the running of a REXX program. The REXX program may be amended to take a variety of actions.

Attention: If you plan to use the SA/390 “bridge” to Tivoli Workload Scheduler, you should be aware that this process uses a version of EQQUX007 which is a driver program that will attempt to call, in turn, modules called UX007001 through to UX007010. Therefore you should use one of these (unused) names for your exit code.

6.1.9 EQQUX009 - the operation initiation exit

Tivoli Workload Scheduler for z/OS has supplied trackers for OS/390® systems and via the end-to-end feature to several other platforms, such as UNIX®. However, if you wish to schedule batch, controlled by Tivoli Workload Scheduler, on an unsupported platform, it is possible to write your own code to handle this.

This exit is called when an operation is ready to start, and uses a workstation that has been defined in the Controller with a USER destination.

Several EQQUX009 samples are supplied with Tivoli Workload Scheduler, one for each of the following operating systems:

- VM, using NJE (EQQUX9N)
- OS/2®, using TCP/IP (EQQX9OS2)
- AIX, using TCP/IP (EQQX9AIX)

6.1.10 EQQUX011 - the job tracking log write exit

This can be used to write a copy of some (or all) Controller events. It passes them to a process that will maintain a job-tracking log copy at a remote site, that may be treated as a disaster recovery site.

The EQQUX011 sample provided by Tivoli Workload Scheduler for z/OS describes a scenario for setting up an effective disaster recovery procedure.

6.2 EQQaaaaa exits

These exits are called by other processes around Tivoli Workload Scheduler; for example, EQQUXCAT is called by the EQQDELDS sample.

Table 6-2 on page 159 lists these and the sample or function that uses them.

Table 6-2 EQQaaaaa Exits

Exit	Description	Used by
EQQUXCAT	EQQDELDS/EQQCLEAN Catalog exit	EQQDELDS sample or EQQCLEAN program
EQQDPUE1	Daily Planning Report	Daily Planning job
EQQUXPIF	Validation of changes done in AD	Job Scheduling Console
EQQUXGDG	EQQCLEAN GDG resolution exit	EQQCLEAN program

6.2.1 EQQUXCAT - EQQDELDS/EQQCLEAN catalog exit

The exit can be used to prevent the EQQDELDS or EQQCLEAN programs from deleting specific data sets.

EQQDELDS is provided to tidy up data sets that would cause a not-cataloged 2 error. It is sometimes inserted in jobs by the EQQUX001 or EQQUX002 exits.

EQQCLEAN is called by the Restart & Cleanup function of Tivoli Workload Scheduler for z/OS when data set cleanup is required before a step restart or rerun of a job.

The sample EQQUXCAT provided by Tivoli Workload Scheduler for z/OS checks the data set name to be deleted and prevents deletion if it starts with SYS1.MAC.

6.2.2 EQQDPUE1 - daily planning report exit

Called by the daily planning batch job, this exit enables manipulation of some of the lines in some of the reports.

6.2.3 EQQUXPIF - AD change validation exit

This exit can be called by the Server or PIF during INSERT or REPLACE AD action.

The sample EQQUXPIF exit supplied by Tivoli Workload Scheduler for z/OS is a dummy that needs validation code building, depending on your requirements.

6.2.4 EQQUXGDG - EQQCLEAN GDG resolution exit

This exit prevents EQQCLEAN from simulating a GDG data set when setting up a job for restart or rerun. This means it does not cause the job to reuse the GDG data set used previously, but allows it to roll the GDG forward as normal.

It does not prevent data set deletion. To do this, use the EQQUXCAT exit. To prevent this process from causing data sets to get out of sync, the checks for both exits should be the same.

The sample EQQUXGDG exit provided by Tivoli Workload Scheduler for z/OS makes several checks, preventing simulation when the job name is MYJOB and the DD name is NOSIMDD, or if the job name is OPCDEMOS and the DDNAME is NOSIMGDG, or the data set name starts with TST.GDG.

6.3 User-defined exits

These exits do not have any prescribed names. They are called when a specific function is invoked that enables the use of an exit to further enhance the capabilities of Tivoli Workload Scheduler.

Table 6-3 lists the function that can call them and the particular activity within that function.

Table 6-3 User-defined exits

Description	Function/activity	Example
JCL imbed	The FETCH directive	//*%OPC FETCH EXIT=program name
Variable substitution	JCL variable	value in subst. exit column in table
Automatic Job Recovery	RECOVER statement	//*%OPC RECOVER CALLEXIT(program name)

6.3.1 JCL imbed exit

This exit is called when the Tivoli Workload Scheduler for z/OS FETCH directive is used and points to an program module instead of a JCL member. It is used to insert additional JCL at that point in the JCL.

6.3.2 Variable substitution exit

This exit is used to provide a value for a variable.

Tivoli Workload Scheduler for z/OS provides a variable substitution exit sample called EQQJVXIT. It provides the possibility to use any value available to Tivoli Workload Scheduler for z/OS regarding the operation, application, or workstation. A few of these are actually provided in the exit code, but you can easily expand the list.

6.3.3 Automatic recovery exit

The automatic recovery exit is called when an error in a job matches a RECOVER directive in the job and the recovery action is to call an exit.

The exit is passed in each line of JCL in the job, which may then be manipulated, deleted or left unchanged. Additionally the exit may also insert lines into the JCL.

The exit can also prevent the recovery taking place.

7

Tivoli Workload Scheduler for z/OS security

Tivoli Workload Scheduler for z/OS security is a complex issue. In this chapter we introduce the basics and give examples of how to set up user profiles. This chapter is not a comprehensive coverage of Tivoli Workload Scheduler for z/OS security. For more about security, refer to *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264, and *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265.

The following topics are covered in this chapter:

- Authorizing the started tasks
- Authorizing Tivoli Workload Scheduler for z/OS to access JES
- UserID on job submission
- Defining ISPF user access to fixed resources

7.1 Authorizing the started tasks

Each started task for Tivoli Workload Scheduler for z/OS is required to have an RACF user ID so the started task can be initiated. The easiest way to do this is to do an ADDUSER command for user ID TWSAPPL, rdefine to the STARTED class for all Tivoli Workload Scheduler for z/OS started tasks using the prefix of TWS*, and associate it with a user of TWSAPPL and group name of stctask.

```
ADDUSER TWSAPPL NAME('TWS Userid') DFLTGRP(stctask) Owner(stctask)
NOPASSWORD
RDEFINE STARTED TWS*.* UACC(NONE) STDATA((USER=TWSAPPL))
```

Note: stctask is a groupname of your installation's choosing that fits your naming standard.

7.1.1 Authorizing Tivoli Workload Scheduler for z/OS to access JES

Tivoli Workload Scheduler for z/OS issues commands directly to JES, so it must have authority to issue commands, submit jobs, and access the JES spool. Perform this only if you are currently restricting this access. If you are allowing this access currently, skip this section. Use Example 7-1 to set up those commands.

Example 7-1 RACF commands

```
RDEFINE JESJOBS SUBMIT.*.*.* UACC(NONE)
RDEFINE OPERCMDS JES2.* UACC(NONE)
RDEFINE OPERCMDS MVS.* UACC(NONE)
RDEFINE JESSPOOL *.* UACC(NONE)
```

The RDEFINE command defines the profile. You must issue a PERMIT command for each of the RDEFINE commands that are issued (Example 7-2).

Example 7-2 Permit commands

```
PERMIT SUBMIT.*.*.* CLASS(JESJOBS) ID(TWSAPPL) ACC(READ)
PERMIT JES2.* CLASS(OPERCMDS) ID(TWSAPPL) ACC(READ)
PERMIT MVS.* CLASS(OPERCMDS) ID(TWSAPPL) ACC(READ)
PERMIT *.* CLASS(JESSPOOL) ID(TWSAPPL) ACC(READ)
```

7.2 UserID on job submission

Tivoli Workload Scheduler for z/OS submits production jobs to the internal reader, or starts started tasks, when all prerequisites are fulfilled. The JCL comes from the JS file (EQQJSnDS) or the JCL job library (EQQJBLIB).

You can determine the authority given to a job or started task in several ways:

- You can submit work with the authority of the Tivoli Workload Scheduler for z/OS address space (as a surrogate). The job or started task is given the same authority as the Controller or Tracker whose submit subtask actually submits the work. For example, work that is transmitted from the Controller and then submitted by the Tracker is given the authority of the Tracker.
- Another method is to use the job submit exit, EQQUX001. This exit is called when Tivoli Workload Scheduler for z/OS is about to submit work.
  - You can use the RUSER parameter of the EQQUX001 exit to cause the job or started task to be submitted with a specified user ID. The RUSER name is supported even if the job or started task is first sent to a Tracker before being started.
  - In certain circumstances you might need to include a password in the JCL to propagate the authority of a particular user. You can use the job-submit exit (EQQUX001) to modify the JCL at submission time and include a password. The JCL is saved in the JCL repository (JSn) data set before the exit is called, thus avoiding the need to store JCL with specific passwords. This method prevents the password from being visible externally. Refer to *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265, for more information about the job-submit exit.
  - If using UX0001, and a user ID is inserted by UX0001, and there is a user ID hardcoded on the jobcard, the job will be submitted as the USERID on the jobcard because the JES reader interpreter will submit the job with the user ID on the jobcard. If this condition occurs and you want to use the user ID that is inserted by UX0001, then you must remove the user ID hardcoded in the jobcard in UX0001.

Refer to Chapter 6, “Tivoli Workload Scheduler for z/OS exits” on page 153 for details on the exits and their use.

7.3 Defining ISPF user access to fixed resources

The AUTHDEF parameter statement (in Tivoli Workload Scheduler for z/OS Controller parms) specifies the fixed resources and subresources that are

passed to RACF with an SAF (system authorization facility) call. When a user accesses the application database and the ad.group is specified in the authdef parameter in parmlib, RACF will check whether the user has access to that group. Refer to Example 7-3.

Example 7-3 Tivoli Workload Scheduler for z/OS Parm Authdef

```
AUTHDEF CLASS(OPCCLASS)  SUBRESOURCES(AD.ADNAME
                                      AD.ADGDDEF
                                      AD.GROUP
                                      AD.JOBNAME
                                      AD.NAME
                                      AD.OWNER
                                      CL.CALNAME
                                      CP.ADNAME
                                      CP.CPGDDEF
                                      CP.GROUP
                                      CP.JOBNAME
                                      CP.NAME
                                      CP.OWNER
                                      CP.WSNAME
                                      CP.ZWSOPER
                                      ET.ADNAME
                                      ET.ETNAME
                                      JS.ADNAME
                                      JS.GROUP
                                      JS.JOBNAME
                                      JS.OWNER
                                      JS.WSNAME
                                      JV.OWNER
                                      JV.TABNAME
                                      LT.ADNAME
                                      LT.LTGDDEF
                                      LT.OWNER
                                      OI.ADNAME
                                      PR.PERNAME
                                      RD.RDNAME
                                      RL.ADNAME
                                      RL.GROUP
                                      RL.OWNER
                                      RL.WSNAME
                                      RL.WSSTAT
                                      SR.SRNAME
                                      WS.WSNAME).
```

It is important to note that the subresource name and the RACF resource name are not the same. You specify the subresource name (such as AD.GROUP) on the authdef statement. The corresponding RACF resource name would be ADG.name (group name) and must be defined in the general resource class used by Tivoli Workload Scheduler for z/OS (see Table 7-1).

Table 7-1 Protected fixed resources and subresources

Fixed resource	Subresource	RACF resource name	Description
AD		AD	Application description file
	AD.ADNAME	ADA.name	Application name
	AD.ADGDDEF	ADD.name	Group-definition -ID name
	AD.NAME	ADN.name	Operator extended name in application description
	AD.OWNER	ADO.name	Owner ID
	AD.GROUP	ADDG.name	Authority group ID
	AD.JOBNAME	ADJ.name	Operation job name in application description
CL		CL	Calendar data
	CL.CALNAME	CLC.name	Calendar name
CP		CP	Current plan file
	CP.ADNAME	CPA.name	Occurrence name
	CP.CPGDDEF	CPD.name	Occurrence group-definition-ID
	CP.NAME	CPN.name	Operation extended name
	CP.OWNER	CPO.name	Occurrence owner ID
	CP.GROUP	CPG.name	Occurrence authority group ID
	CP.JOBNAME	CPJ.name	Occurrence operation name
	CP.WSNAME	CPW.name	Current plan workstation name
	CP.ZWSOPER	CPZ.name	Workstation name used by an operation
ETT		ETT	ETT dialog
	ET.ETNAME	ETE.name	Name of triggering event
	ET.ADNAME	ETA.name	Name of application to be added
JS		JS	JCL and job library file
	JS.ADNAME	JSA.name	Occurrence name
	JS.OWNER	JSO.name	Occurrence owner ID
	JS.GROUP	JSG.name	Occurrence authority group ID
	JS.JONAME	JSJ.name	Occurrence operation name
	JS.WSNAME	JSW.name	Current plan works at ion name
JV		JV	JCL variable-definition file
	JV.OWNER	JBO.name	Name of JCL variable table

Fixed resource	Subresource	RACF resource name	Description
LT	 LT.ADNAME LT.LTGDDEF LT.OWNER	LT LTA.name LTD.name LTO.name	Long-term plan file Occurrence name Occurrence group definition ID Occurrence owner ID
OI	 OI.ADNAME	OI.name	Operator instruction file Application name
PR	 PR.PERNAME	PR PRP.name	Period data Period name
RL	 RL.ADNAME RL.OWNER RL.GROUP RL.WSNAME RL.WSSTATr	RL RLA.name RLO.name RLG.name RLW.name RLX.name	Ready list data Occurrence name Occurrence owner ID Occurrence authority group ID Current plan workstation name Current plan workstation changed by WSSTAT
RD	 RD.RDNAME	RD RDR.name	Special resource file Special resource name
SR	 SR.SRNAME	SR SRS.name	Special resources in the current plan Special resource name
WS	 WS.WSNAME	WS WSW.name	Workstation data Workstation name in workstation database
ARC		ARC	
BKP		BKP	
CMAC		CMAC	Clean action
CONT		CONT	Refresh RACF subresources
ETAC		ETAC	
EXEC		EXEC	EX (executes) row command
JSUB		JSUB	Activates/deactivates job submission
REFR		REFR	Refresh LTP and delete CP
WSCL		WSCL	All workstation closed data

Here are some notes on the fixed resources and subresources:

- The AD.JOBNAME and CP.JOBNAME subresources protect only the JOBNAME field within an application or occurrence. You use these

subresources to limit the job names that a user has access to during job setup and similar tasks. If you do not use these subresources, a dialog user might obtain greater authority by using Tivoli Workload Scheduler for z/OS to perform certain functions. For example, a user could submit an unauthorized job by adding an application to the current plan, changing the job name, and then letting Tivoli Workload Scheduler for z/OS submit the job. For these subresources, only the ACCESS(UPDATE) level is meaningful.

- The subresources AD.GROUP, CP.GROUP, JS.GROUP, and RL.GROUP are used to protect access to Tivoli Workload Scheduler for z/OS data based on the authority group ID and not application description groups.
- The subresource data is passed to SAF without modifications. Your security product might have restrictions on which characters it allows. For example, RACF resource names cannot contain asterisks, embedded blanks, or DBCS characters.
- The EQQ9RFDE member in the sample library updates the class-descriptor tables with a Tivoli Workload Scheduler-specific class, called OPCCLASS.
- Use the CP.ZWSOPER subresource if you want to protect an operation based on the name of the workstation where the operation will be started. You must have update access to this subresource if you want to modify an operation. If you want to specify dependencies between operations, you must have update authority to both the predecessor and successor operations. You can use the CP.ZWSOPER subresource to protect against updates to an operation in an occurrence or the unauthorized deletion or addition of an operation in an occurrence. This subresource is not used to protect the addition of an occurrence to the current plan or to protect an occurrence in the current plan that a user attempts to delete, set to waiting, or set to complete. When an occurrence is rerun, access authority is checked only for the particular operation that the rerun is started from. The subresource CP.ZWSOPER is unlike the subresource CP.WSNAME, which protects workstations but does not protect against updates to operations.
- When no current plan occurrence information is available, subresource protection for job setup and JCL editing tasks is based on information from the application description. For example, if you are adding an occurrence to the CP and you request JCL edit for an operation, subresource requests using owner ID or authority group ID are issued using the owner ID or authority group ID defined in the AD, because the CP occurrence does not yet exist. Similarly, when editing JCL in the LTP dialog, subresources are based on CP occurrence information if the occurrence is in the CP. If the occurrence is not in the CP, subresource requests are issued using information from the AD.

The following list gives some of the RACF fixed resources and their usage:

ARC The ACTIVATE/DEACTIVATE automatic recovery function in the Tivoli Workload Scheduler for z/OS Service Functions dialog. To use this function, you need update authority to the ARC fixed resource.

BKP The use of the BACKUP command. BACKUP lets you request a backup of the current plan data set or JCL repository data set. To use this command, you need to update access to the BKP fixed resource on the system where the command is issued.

CMAC The Catalog Management Actions fixed resource that can be used to control catalog cleanup actions. To start the cleanup actions you must update authority to the CMAC fixed resources.

CONT The RACF RESOURCES function in the Tivoli Workload Scheduler for z/OS Service Functions dialog. This lets you activate subresources that are defined after Tivoli Workload Scheduler for z/OS is started. To use this function, you need update authority to the CONT fixed resource.

ETAC The ACTIVATE/DEACTIVATE ETT function in the Service Functions dialog. To use this function, you need update authority to the ETAC fixed resource.

EXEC The use of the EX (execute) row command. You can issue this command from the Modify Current Plan dialog and workstation ready lists, if you have update access to the EXEC fixed resource.

JSUB The ACTIVATE/DEACTIVATE job submission function in the Tivoli Workload Scheduler for z/OS Service Functions dialog or TSO JSUACT command. To use this function, you need update authority to the JSUB fixed resource.

REFR The REFRESH function (delete current plan and reset long-term plan) in the Tivoli Workload Scheduler for z/OS Service Functions dialog. To use this function, you need update authority to the REFR fixed resource.

WSCL The All Workstations Closed function of the Workstation Description dialog. To browse the list of time intervals when all workstations are closed requires read authority to the WSCL fixed resource. Updating the list requires update authority to the WSCL fixed resource.

Important: Ensure that you restrict access to these fixed resources to users who require them. REFR is particularly important because this function deletes the current plan.

7.3.1 Group profiles

Instead of defining each user to Tivoli Workload Scheduler for z/OS resources it makes sense to define group profiles and give each user access to a specific group profile. You might define four different groups (for example, an operator group, a scheduler group, a special group for a group of application analysts that might require just a few applications, and a group for the support group such as administrative users and system programmers).

Before beginning the defining of group profiles look at Table 7-2, which shows what resources are required for each function.

Table 7-2 Resources required for each function

DIALOG	FUNCTION	FIXED RESOURCE	ACCESS TYPE
Work station	Browse workstation	WS	Read
	Update workstation	WS WSCL	Update Read[1]
	Browse workstation closed	WSCL	Read
	Update workstation closed	WSCL	Update
	Print	None	None
Calendar	Browse	CL	Read
Update	CL	Update	
Print	None	None	
Period	Browse	PR	Read
	Update	PR JV	Update Read[2]
	Print	CL	Read

DIALOG	FUNCTION	FIXED RESOURCE	ACCESS TYPE
Application Description	Browse	AD CL WS OI RD	Read Read Read Read[3] Read[13]
	Update	AD CL PR WS OI JV RD	Update Read Read Read Update[4] Read[2] Read[14]
	Print	WS	Read[5]
	Mass update	AD CL PR WS JV RD	Update Read Read Read Read Read[14]
Operator Instruction	Browse	OI	Read
	Update	OI	Update
	Print	None	None
	Mass update	None	None
Special Resource	Browse	RD WS	Read Read
	Update	RD WS	Update Read
Event Triggered	Browse	ETT	Read
	Update	ETT	Update

DIALOG	FUNCTION	FIXED RESOURCE	ACCESS TYPE
Job Description	Browse	AD WS OI RD	Read Read Read[3] Read[13]
	Update	AD CL PR WS OI JV RD	Update Read Read Read Update[4] Read[2] Read[14]
	Print	WS	Read
JCL Variable Table	Browse	JV	Read
	Update	JV	Update
	Print	JV	Read
Long-term plan	Browse	LT AD CL PR WS	Read Read Read Read Read
	Update (delete or modify) or add	LT AD CL PR WS JV	Update Read Read Read Read Read[2]
	Job setup	LT AD CL PR WS JS	Read Read Read Read Read Update
	Batch	LT	Read
	Display Status	LT	Read
	Set defaults	None	None
Daily Planning	Batch	CP	Read

DIALOG	FUNCTION	FIXED RESOURCE	ACCESS TYPE
Work Station Communication	Using ready lists	RL CP JS OI JV EXEC	Update[6] Read[7] Update[8] Read[9] Read[10] Update[12]
	Waiting list	CP JS OI	Read Update[8] Read9
	Job Setup	CP JS OI	Read Update Read[9]
	Review workstation status	CP	Read
	Define ready lists	None	None
Modify Current Plan	Add	AD CP JS JV SR	Read Update Read Read[2] Update[15]
	Update (delete or modify), change status of workstation	CP JS JV SR	Update Update[8] Read[2] Update[15]
	Change status, rerun, error handling	CP JS OI EXEC	Update Update[8] Read[9] Update[12]
	Restart and cleanup	CP JS CMAC	Update Update Update
	Browse	CP JS OI SR	Read Read[11] Read[9] Read[3]
	Job setup	CP JS	Read Update[8]
	Define error lists	None	None

DIALOG	FUNCTION	FIXED RESOURCE	ACCESS TYPE
Query current plan	All	CP JS OI SR	Read Read[11] Read[9] Read[13]
Service Functions	Activate/deactivate job submission	JSUB CP	Update Update
	Activate/deactivate automatic recovery	ARC CP	Update Update
	Refresh (delete current plan and reset long-term plan)	REFR LT	Update Update
	Activate RACF resources	CONT	Update
	Activate/deactivate event-triggered tracking	ETAC	Update
	Produce APAR tape	None	None

Table notes:

1. If you are modifying open intervals for one day
2. If you specify or update a JCL variable table name
3. If you are browsing operator instructions
4. If you are modifying operator instructions
5. If sorted in workstation order
6. If you want to change status
7. If you request a review of details
8. If you want to modify JCL
9. If you want to browse operator instructions
10. If you perform job setup using JCL variable substitution
11. If you want to browse JCL
12. If you want to issue the EX (execute) command
13. If you want to browse special resources
14. If you want to specify special resource allocations
15. If you want to add or update special resources

Figure 7-1 shows an example of an operator group profile that does not include update access to the application database, calendars, and variables.

Fixed Resource	Sub Resource	Authority	Resource
AD	AD.*	READ	Application database
CL	CL.*	READ	Calendars
CP	CP.*	UPDATE	Current Plan
ETT	ETT.*	READ	ETT Dialog
JS	JS.*	UPDATE	JCL and Job library
JV	JV.*	READ	JCL Variable Definition
LT	LT.*	UPADATE	Long Term Plan
OI	OI.*	READ	Operator Instructions
PR	PR.*	READ	Periods
RL	RL.*	UPDATE	Ready List
RD	RD.*	READ	Special Resource File
SR	SR.*	UPDATE	Special Resources in Current plan
WS	WS..*	UPDATE	Work Station
ARC		READ	Activate Auto Recovery
BKP		READ	Backup Command
CMAC		UPDATE	Clean Up Action
CONT		READ	Refresh RACF
ETAC		READ	Activate ETT
EXEC		UPDATE	Issue Row Commands
JSUB		READ	Activate Job Submission
REFR		READ	Refresh Long Term Plan and delete Current Plan
WSCL		READ	All Work Station Closed Data

Figure 7-1 Operator group profile

An RACF profile for a scheduler, as shown in Figure 7-2, might allow much more access. It also might not allow access to a certain application. Note when using subresources that there should be a good naming standard for the applications, or the profile can become cumbersome. An example is a set of applications all starting with PAYR so in the subresource you might use ad.payr*, so all applications with that HLQ (high-level-qualifier) would be used. You can see that if you used PAY, HRPAY, R1PYROLL, the HLQ could become cumbersome in the group profile. Also in the example, note that the scheduler is restricted from update for the current plan, JCL, and JCL variables for an application called PAYR*.

Fixed Resource	Sub Resource	Authority	Resource
AD	AD.*	UPDATE	Application database
CL	CL.*	UPDATE	Calendars
CP	CP.* CPA.PAYR*	UPDATE READ	Current Plan
ETT	ETT.*	UPDATE	ETT Dialog
JS	JS.* JSA.PAYR*	UPDATE READ	JCL and Job library
JV	JV.* JVO.PAYR*	UPDATE READ	JCL Variable Definition
LT	LT.*	UPDATE	Long Term Plan
OI	OI.*	UPDATE	Operator Instructions
PR	PR.*	UPDATE	Periods
RL	RL.*	UPDATE	Ready List
RD	RD.*	UPDATE	Special Resource File
SR	SR.*	UPDATE	Special Resources in Current plan
WS	WS.*	UPDATE	Work Station
ARC		READ	Activate Auto Recovery
BKP		UPDATE	Backup Command
CMAC		UPDATE	Clean Up Action
CONT		UPDATE	Refresh RACF
ETAC		UPDATE	Activate ETT
EXEC		UPDATE	Issue Row Commands
JSUB		UPDATE	Activate Job Submission
REFR		READ	Refresh Long Term Plan and delete Current Plan
WSCL		UPDATE	All Work Station Closed Data

Figure 7-2 Scheduler group profile

A profile for application analyst group (such as Figure 7-3) might be very restrictive so that the only applications that they would have access to modify would be their own (all applications starting with ACCT). To do this would require the use of subresources, HLQs, and permitting the group access to those HLQs.

Fixed Resource	Sub Resource	Authority	Resource
AD	AD.*	READ	Application database
CL	CL.*	READ	Calendars
CP	CP.* CPA.ACCT*	READ UPDATE	Current Plan
ETT	ETT.* ETA.ACCT*	READ UPDATE	ETT Dialog
JS	JS.* JSA.ACCT*	READ UPDATE	JCL and Job library
JV	JV.* JVO.ACCT*	READ UPDATE	JCL Variable Definition
LT	LT.*	READ	Long Term Plan
OI	OI.* OIA.ACCT*	READ UPDATE	Operator Instructions
PR	PR.*	READ	Periods
RL	RL.* RLA.ACCT*	READ UPDATE	Ready List
RD	RD.*	READ	Special Resource File
SR	SR.* SRS.ACCT*	READ UPDATE	Special Resources in Current plan
WS	WS..*	READ	Work Station
ARC		READ	Activate Auto Recovery
BKP		READ	Backup Command
CMAC		READ	Clean Up Action
CONT		READ	Refresh RACF
ETAC		READ	Activate ETT
EXEC		UPDATE	Issue Row Commands
JSUB		READ	Activate Job Submission
REFR		READ	Refresh Long Term Plan and delete Current Plan
WSCL		READ	All Work Station Closed Data

Figure 7-3 Application analyst profile

The profile shown in Figure 7-4 is the least restrictive profile, it has no restricted resources that the Administrator or System Programmer can use in Tivoli Workload Scheduler. This profile should be given to only a select few people because it allows all access.

Fixed Resource	Sub Resource	Authority	Resource
AD	AD.*	UPDATE	Application database
CL	CL.*	UPDATE	Calendars
CP	CP.*	UPDATE	Current Plan
ETT	ETT.*	UPDATE	ETT Dialog
JS	JS.*	UPDATE	JCL and Job library
JV	JV.*	UPDATE	JCL Variable Definition
LT	LT.*	UPDATE	Long Term Plan
OI	OI.*	UPDATE	Operator Instructions
PR	PR.*	UPDATE	Periods
RL	RL.*	UPDATE	Ready List
RD	RD.*	UPDATE	Special Resource File
SR	SR.*	UPDATE	Special Resources in Current plan
WS	WS.*	UPDATE	Work Station
ARC		UPDATE	Activate Auto Recovery
BKP		UPDATE	Backup Command
CMAC		UPDATE	Clean Up Action
CONT		UPDATE	Refresh RACF
ETAC		UPDATE	Activate ETT
EXEC		UPDATE	Issue Row Commands
JSUB		UPDATE	Activate Job Submission
REFR		UPDATE	Refresh Long Term Plan and delete Current Plan
WSCL		UPDATE	All Work Station Closed Data

Figure 7-4 Administrator group profile

8

Tivoli Workload Scheduler for z/OS Restart and Cleanup

This chapter discusses the many features of the Tivoli Workload Scheduler for z/OS Restart and Cleanup option. We cover and illustrate quite a few aspects to give a wider view of what happens during a restart. This chapter also covers all commands that are part of the Restart and Cleanup option in Tivoli Workload Scheduler for z/OS.

This chapter has the following sections[1]:

- Implementation
- Cleanup Check option
- Ended in Error List criteria
- Steps that are not restartable

[1] Some of the content in this chapter was provided by Rick Marchant

8.1 Implementation

DataStore is an additional started task that is required for Restart and Cleanup. DataStore collects a local copy of the sysout data for Tivoli Workload Scheduler for z/OS jobs and, only when a user requests a copy for a restart or a browse of the joblog, sends a copy to the Tivoli Workload Scheduler for z/OS Controller. (For more information about DataStore, see Chapter 1, “Tivoli Workload Scheduler for z/OS installation” on page 3.) There is one DataStore per JES spool. DataStore (started task) is an additional required component. Other additional parameters for the Controller include: RCLOPTS, FLOPTS, and OPCOPTS. For DataStore, there is DSTOPTS; for Batchopt, there is RCLEANUP.

Important: An extremely important initialization parameter, which is in the RCLOPTS, is STEPRECHCK(NO). This parameter gives the Tivoli Workload Scheduler for z/OS user the ability to point to a specific step at restart, even if the step has been set as a non-restartable one. It is very important to make this change. Based on our prior experiences, customers that had *not* added STEPRESCHCK(NO) seemed to run into some problems when trying to do step restarts.

8.1.1 Controller Init parameters

Example 8-1 on page 183 shows OPCOPTS used to enable Restart and Cleanup. It is recommended to use FLOPTS, which provides DataStore options for XCF connectivity.

Have your systems programmer set up the RCLOPTS - STEPRECHCK(NO) parameter. As previously mentioned, this is extremely important.

Batchopt is used when the current plan extends. This will clean up any of the Controller SDF files that DataStore has collected.

Example 8-1 Controller Init parameters

```
OPCOPTS
      RCLEANUP(YES)             /* Enable Restart/Cleanup*/

/************** USE THE FOLLOWING FOR  ** XCF ** CONNECTIVITY****************/

FLOPTS                            /* Datastore options    */
       XCFDEST(tracker.dststore)  /* TRACKER - DATASTORE XCF   */
       DSTGROUP(dstgroup)          /* DATASTORE XCF GROUPNAME */
       CTLMEM(xxxxCTL1)       /*  CONTROLLER XCF MEMBERN  */

       /************** USE THE FOLLOWING FOR  ** VTAM ** CONNECTIVITY **********************/

FLOPTS                            /* Datastore options    */
       SNADEST(trackerLU.datastorLU)  /* TRACKER - DATASTORE NCF   */
       CTLLUNAM(cntlDSTLU)      /*  Controller/Datastor LUNAME         */


RCLOPTS
      CLNJOBCARD('OPC')         /* job card - std alone clnup    */
       CLNJOBPX(EQQCL)           /* job name pref-std alone clup  */
       DSTRMM(N)                 /* RMM Interface                 */
      DSTDEST(TWSC)             /* dest - sysout cpy for data str*/
       DSTCLASS(********:X,trkrdest:X) /* JCC/Achiver alt class   */       STEPRESCHK(NO)
   /* Override restart step logic   */
BATCHOPTS
      RCLEANUP(YES)                /*  Enable clnup for contlr SDF files*/
```

Example 8-2 is a sample copy of the Controller Init parameters. This can be customized accordingly for your site.

Example 8-2 Controller Init parameters

```
DATA STORE:

DSTOPTS CINTERVAL(300)          /* Cleanup every 5 hours            */
        CLNPARM(clnmembr)   /* Cleanup rules membername         */
        DELAYTIME(15)           /* Waitime to discard incomplete    */
        DSTLOWJOBID(1)          /* Lowest job number to search      */
        DSTHIGHJOBID(32767)     /* Highest job number to search     */
        FAILDEST(FAILDEST)      /* Requeue dest jobs not archived   */
        HDRJOBNAME(JOBNAME)     /* Jobname ID for stp tbl hdr       */
        HDRSTEPNAME(STEPNAME)  /* Stepname ID for stp tbl hdr      */
        HDRPROCNAME(PROCNAME)    /* Procstep ID for stp tbl hdr      */
        HDRJOBLENGTH(21)        /* Strt pos - jobname in step tbl   */
        HDRSTEPLENGTH(30)       /* Strt pos - stepname - stp tbl hdr*/
        HDRSTEPNOLENGTH(120)    /* Strt pos - stepno - step tbl hdr */
        HDRPROCLENGTH(39)       /* Strt pos - prcname - stp tbl hdr */
```

```
/*
*/
/************** USE THE FOLLOWING FOR VTAM CONNECTIVITY***********************/
        HOSTCON(SNA)             /* Talk to Controller via NCF       */
              CTLLUNAM(cntllunm) /* NCF Controller LUname            */
              DSTLUNAM(dstlunm) /* NCF DataStore LUname             */
/*
*/
/************** USE THE FOLLOWING FOR  ** XCF ** CONNECTIVITY*****************/
        HOSTCON(XCF)             /* Talk to Controller via XCF       */
              CTLMEM(twsmembr)   /* XCF Controller Member            */
              DSTGROUP(dstgroup)  /* XCF DataStore Group              */
              DSTMEM(dstmembr)   /* XCF DataStore Member             */
/*
*/
       MAXSTOL(0)               /* Maxuser - no user sysout         */
       MAXSYSL(0)               /* No. of user sysouts sent to ctlr */
       NWRITER(2)               /* No. of Datastore writer subtasks */
       QTIMEOUT(15)             /* Timeout (minutes) for Joblogretrv*/
       RETRYCOUNTER(1)          /* retry intrval for job archival   */
       STORESTRUCMETHOD(DELAYED) /* only gen/arch sdf info   on req */
       STOUNSD(YES)             /* store udf data / enable jblog retrieval*/
       SYSDEST(TWSC)            /* dest for data store sysout       */
                                /* MUST match RCLOPTS DSTDEST   */
       WINTERVAL(5)             /* Seconds between SYSCLASS scans */
```

8.2 Cleanup Check option

When you select option `0.7` on the main menu (see Figure 8-1 on page 185), the Setting Check for Automatic Cleanup Type panel is displayed:

- Specify Y to check and possibly modify the Cleanup Dataset list, displayed on the Modifying Cleanup Actions panels, even if the Automatic option is specified at operation level.
- Specify N to bypass the check.

In this case, the Confirm Restart panel is directly displayed when you request a RESTART function with the cleanup type Automatic. The default is N. This action is a one-time change and is set according to user ID.

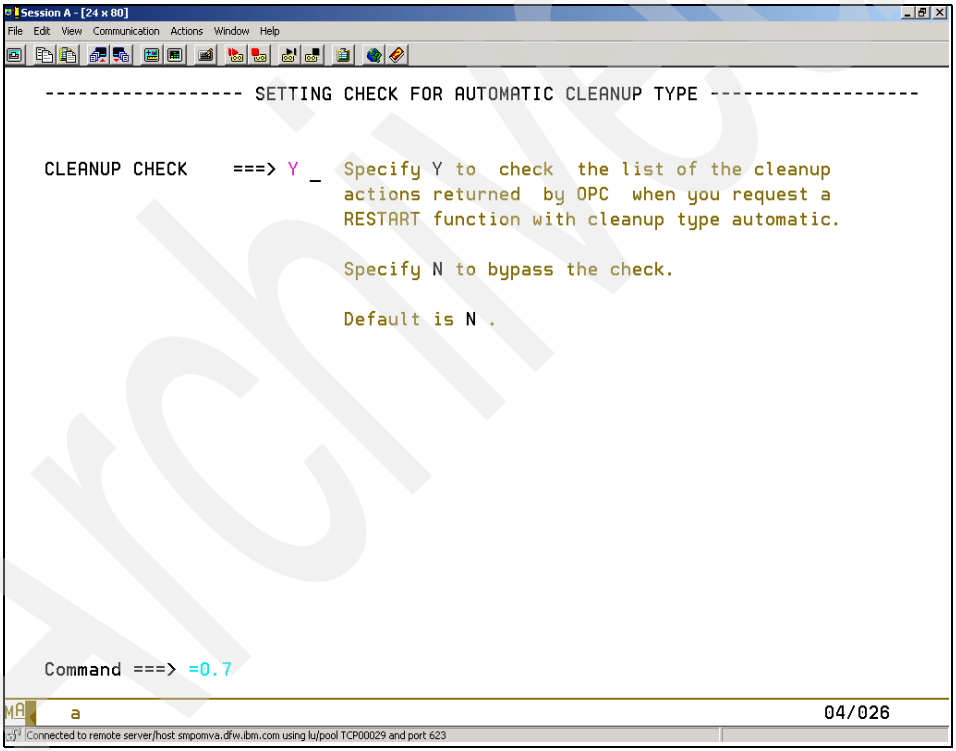


Figure 8-1 Setting Check for Automatic Cleanup Type

8.2.1 Restart and Cleanup options

To set up Restart and Cleanup options in Tivoli Workload Scheduler:

1. In the Tivoli Workload Scheduler for z/OS database, choose option `=1.4.3`.

2. Select the application you would like to modify and type the `OPER` command to take you to the Operations panel.
3. Type an `S.9` row command (see Figure 8-2).

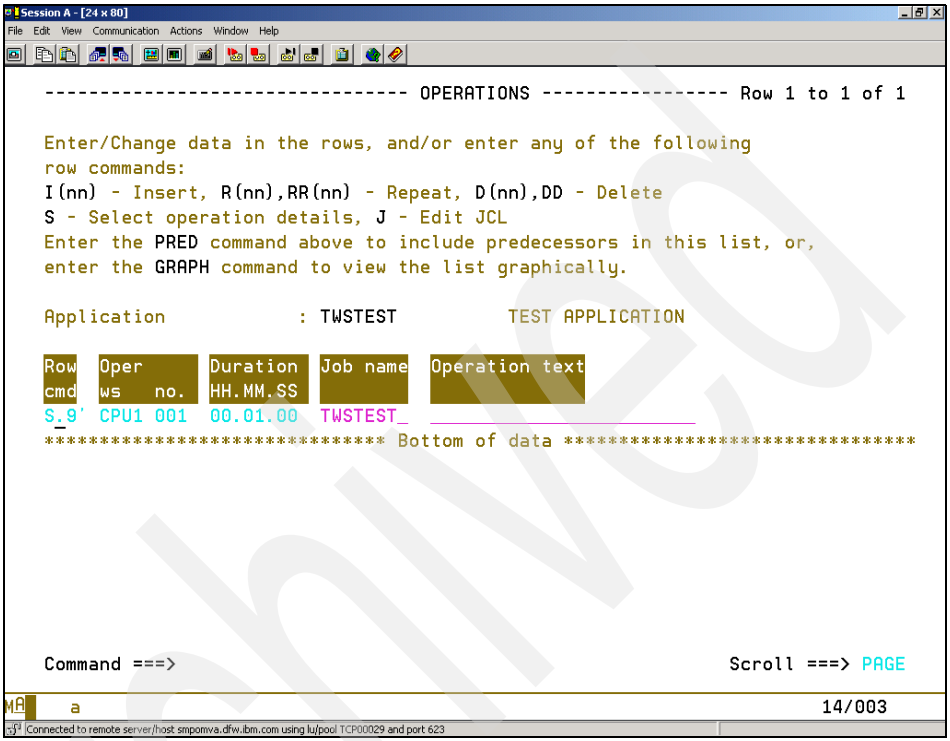


Figure 8-2 Operations from the Tivoli Workload Scheduler for z/OS database

4. You should see the panel shown in Figure 8-3 on page 187, where you can specify the cleanup action to be taken on computer workstations for operations that end in error or are rerun:
   - A (Automatic): With the Automatic setting, the Controller automatically finds the cleanup actions to be taken and inserts them as a first step in the JCL of the restarted job. The cleanup actions are shown to the user for confirmation if the AUTOMATIC CHECK OPC dialog option is set to YES (recommended).
   - I (Immediate): If you set Immediate, the data set cleanup will be performed immediately if the operation ends in error. The operation is treated as if it had the automatic option when it is rerun.
   - M (Manual): If you set Manual, the data set cleanup actions are deferred for the operation. They will be performed when initiated manually from the dialog.

– N (None): None means that the data set cleanup actions are not performed for the operation if it ends in error or is rerun.
– Expanded JCL: Specify whether OPC will use the JCL extracted from JESJCL sysout:
  - Y: JCL and Proc are expanded in Tivoli Workload Scheduler for z/OS at restart.
  - N: Allows alternate JCL and Procs to be inserted.
– User Sysout: Specify whether User sysout support is needed.
  - Y: DataStore logs User sysout too.
  - N: DataStore does not log User sysout.

Figure 8-3 shows the suggested values: Cleanup Type: A, Expanded JCL: N, and User Sysout: N.

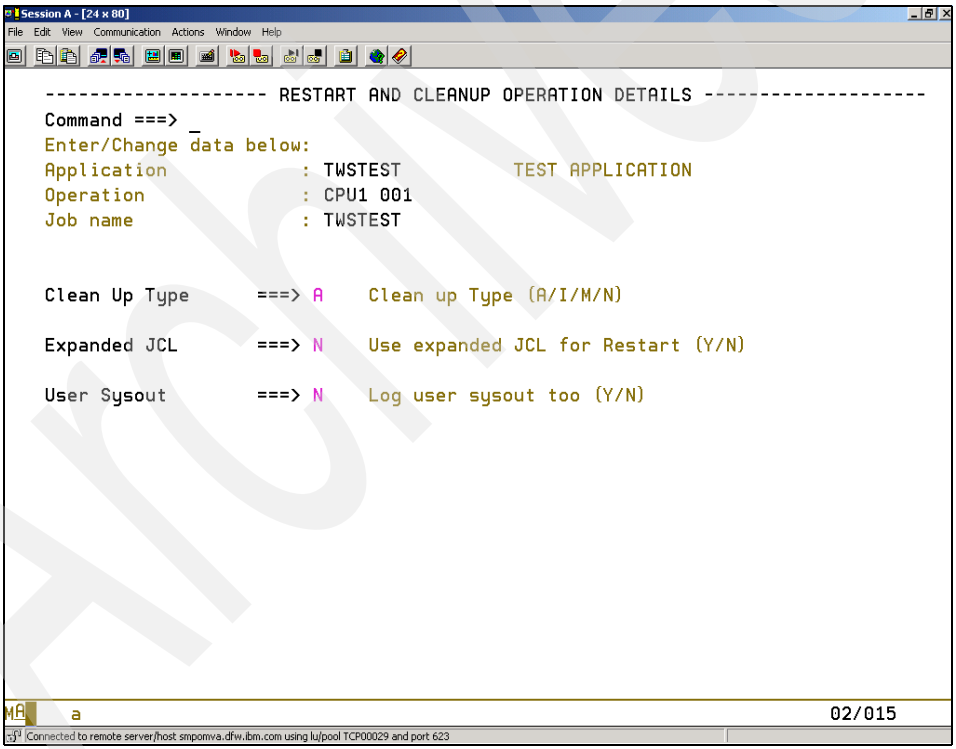


Figure 8-3 Restart and Cleanup Operations Detail menu

8.3 Ended in Error List criteria

We now cover how to perform a restart from the error list.

1. You can access the error list for jobs that ended in error in the current plan by choosing option 5 to modify the current plan, then option 4 Error Handling, or simply =5.4 from most anywhere in Tivoli Workload Scheduler. See Figure 5-4 for the Specifying Ended in Error List Criteria.

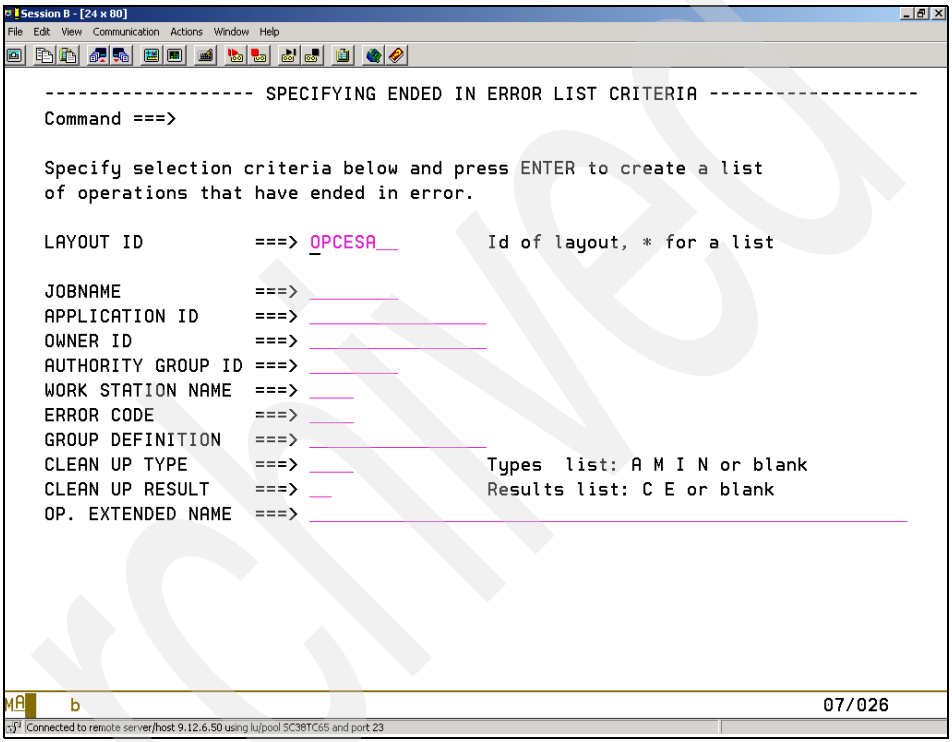


Figure 8-4 Specifying Ended in Error List Criteria option =5.4

The Layout ID is where you can specify the name of a layout to be used or, if left blank, you can select from a list of available layouts. You can also create your own layout ID by going to =5.9. The Jobname field is where you can list just one job, leave the field blank, or insert an asterisk for a listing of all jobs that are on the Error List (Optional).

In the Application ID field, you can specify an application name to get a list of operations in error from that particular application. As with jobname, to get all applications you can leave the field blank or type an asterisk (Optional). You can combine the global search characters (* and %) to get a group of

applications. Using the global search characters can help with both the application and jobname fields.

In the Owner ID field, specify an owner name to get just one owner or to get all owners; again, you can leave it blank, type an asterisk, or combine search characters (* and %) to get a group of owners. For the Authority Group ID you can insert an authority group name to get just one authority group or use the same search scenarios as previously discussed with Owner ID. For the Work Station Name you can insert a specific workstation name or leave the field blank or with an asterisk to get all workstations.

From the Error Code field, you can specify an error code to get all jobs that may have ended with that particular error or leave it blank or use the same search parameters as previously discussed. The same thing applies for Group Definition when doing a search you can specify a group definition ID to obtain the operations in error, which are members of the occurrence group, and to get all leave it blank or insert an asterisk.

For Clean Up Type, you can select one of the following options, or leave it blank to select all statuses:

- A - Automatic
- I - Immediate
- M - Manual
- N - None

You can choose Clean Up Result with one of the following or simply leave the field blank:

- C - Cleanup completed
- E - Cleanup ended in error

Finally, you can enter the extended name of the operation in the OP. Extended Name field.

2. Now we are in the Handling Operations that Ended in Error panel (Figure 8-5 on page 190). Here you will find the jobs that ended in error based on your initial setup in the prior panel by filtering or not filtering out the Error list. Note that your Layout ID is the same as indicated on the panel. The error list shows the date and time the job ended in error, the application with which the job is associated, and the error code.
3. The top of the panel shows several row commands that you can use.
   - C: Set the operation to Completed Status.
   - FJR: Fast path to run Job Restart. Defaults are used.
   - FSR: Fast path to run Step Restart. Defaults are used.
   - I: Browse detailed information for the operation.

- J: Edit JCL for the operation.
- L: Browse the job log for the operation. If the job log is not currently stored but can be obtained, a retrieval request is sent. You are notified when the job log is retrieved.
- MH: Manually hold an operation.
- MR: Release an operation that was manually held.
- O: Browse operator instructions.
- RC: Restart and clean up the operation. If the OperInfo is not available, a retrieval request is sent. You are notified when OperInfo is retrieved.
- SJR: Simple Job Restart. Operation status is set to ready.

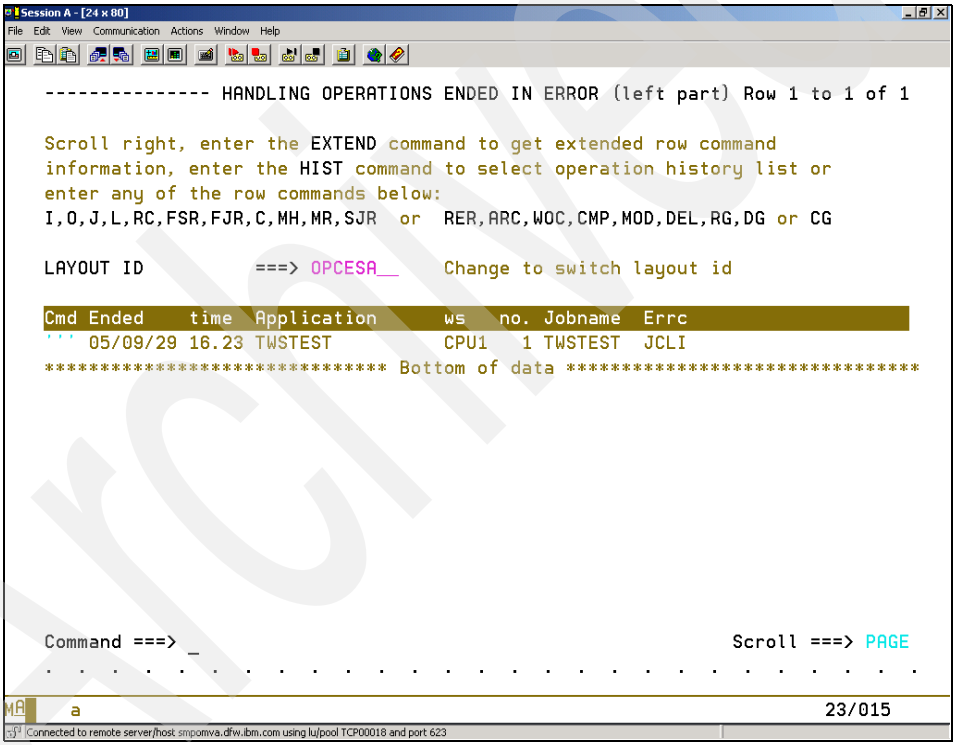


Figure 8-5 Handling Operations that Ended in Error

These commands are available for the occurrence:

- ARC: Attempt Automatic Recovery.
- CMP: Set the status of the occurrence to complete.

– CG: Complete the occurrence group. All occurrences belonging to this group will be complete.
– DEL: Delete the occurrence.
– DG: Delete an occurrence group. All occurrences belonging to this group will be deleted.
– MOD: Modify the occurrence.
– RER: Rerun the occurrence.
– RG: Remove this occurrence from the occurrence group.
– WOC: Set all operations in the occurrence to Waiting Status.

You can also run the EXTEND command to get this detailed list above.

4. For our job that ended in error in Figure 8-5 on page 190, there is a rather simple fix. We see that the ERRC is JCLI, so now we can look at and edit the JCL by designating J on the row command.

   In the JCL in Figure 8-6, we can see the initial problem: The EXEC in the first step is misspelled, so we can change it from EXC to EXEC, then type END or press the F3 key, or if you want to cancel just type CANCEL.

   For our example, we change the spelling and type END to finish, which returns us to Figure 8-5 on page 190.

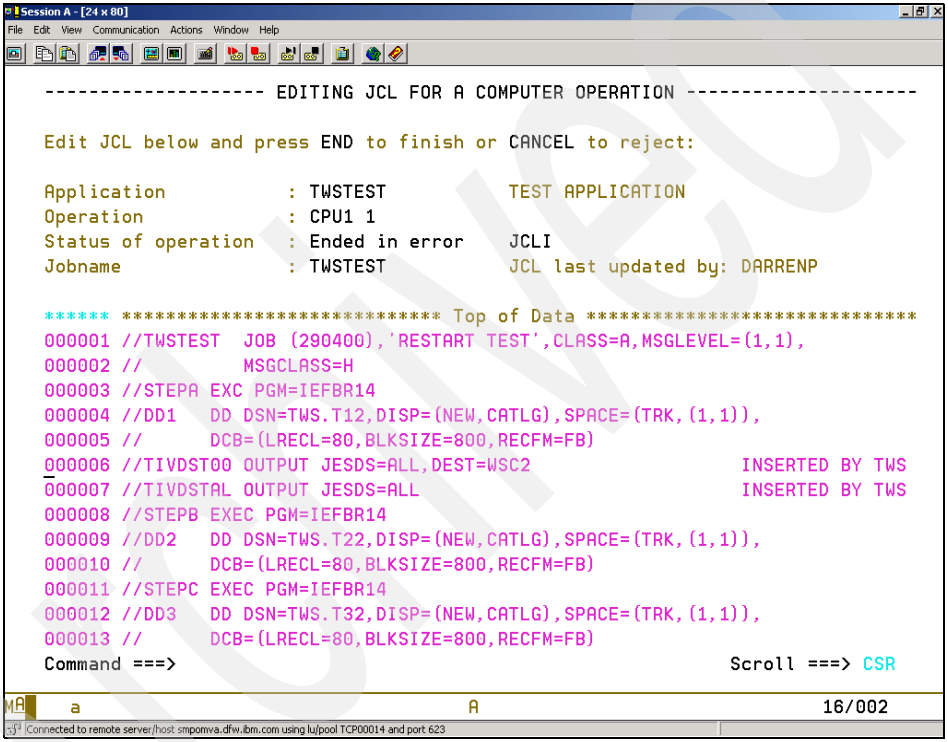


Figure 8-6 Editing JCL for a Computer Operation

5. From here we perform an SJR (Simple Job Restart) on the row command. Because this was a small JCL problem, the job will restart and finish to completion.

6. When you enter the SJR row command, Tivoli Workload Scheduler for z/OS prompts you with the Confirm Restart of an Operation panel (Figure 8-7). Here you can either select Y to confirm the restart or N to reject it. You can enter additional text for the Reason for Restart; when this is entered it goes to the track log for audit purposes.

   After you have typed in your Reason for Restart and then entered Y on the command line, you will be taken back to the Error List (Figure 8-7). From here, you will notice the operation is no longer in the Error List, or if the job got another error it will be back in the error list. Otherwise, the job is submitted to JES and is running as normal.

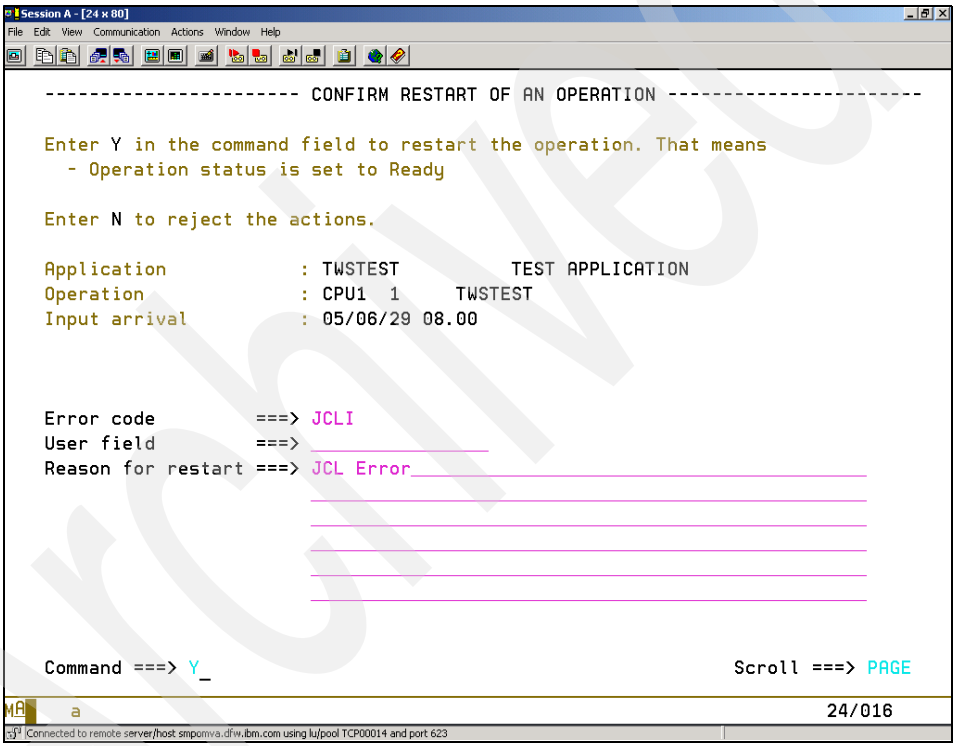


Figure 8-7 Confirm Restart of an Operation

7. In our next example, we look into a simple step restart. Figure 8-8 shows a job in the error list with an ERRC S806.

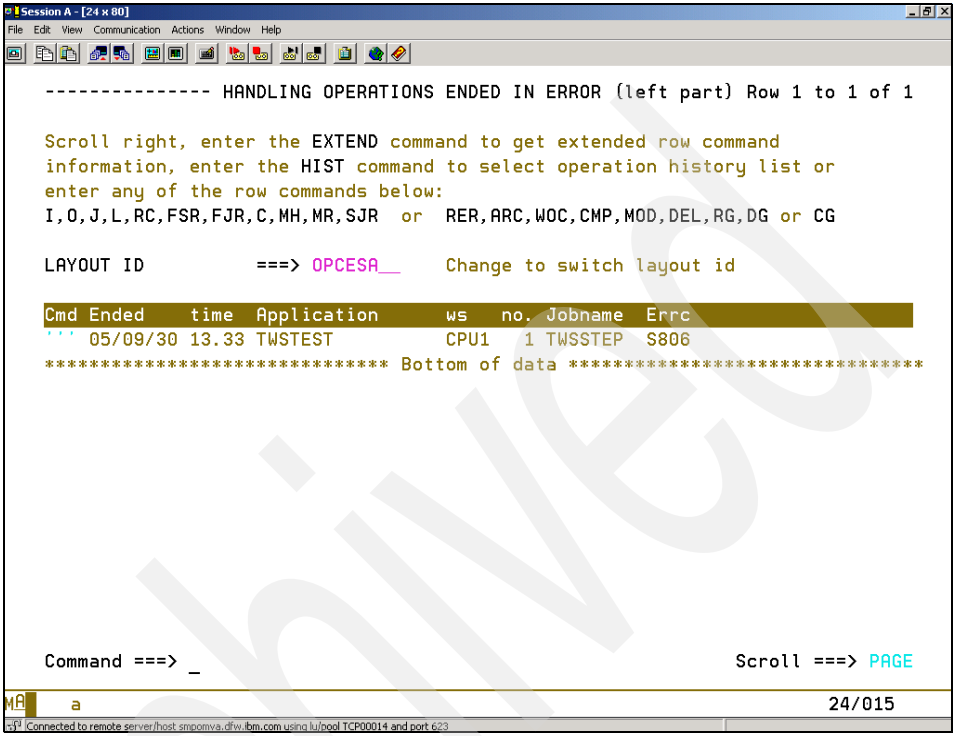


Figure 8-8 Handling Operations that Ended in Error for Step Restart

Our JCL in this operation is in Example 8-3 on page 195. As you can see, STEP1, STEP2, and STEP3 are executed; STEP4 gets an S806, and STEP5 is flushed. Step restart enables you to restart this job at the step level and then performs the necessary cleanup actions. When you request a step restart, Tivoli Workload Scheduler for z/OS shows which steps are restartable and, based on this, it will provide you with the best option. You can override the selection that is made by Tivoli Workload Scheduler.

Step restart works based on the use of the DataStore and the simulation of return codes. Tivoli Workload Scheduler for z/OS adds a preceding step called EQQCLEAN, which does the simulation from the history of the previous runs and it also performs the cleanup action.

The return code simulation will not let you change the JCL structure when performing a step restart. EQQCLEAN uses the list of step names and the return codes associated with them that are provided by Tivoli Workload Scheduler, so that if a simulated step no longer exists in the submitted JCL, the EQQCLEAN program fails with a message about step mismatch. Also,

changing the expanded JCL to NO from YES when it has already restarted using restart and cleanup causes a mismatch between the steps listed in the Step Restart Selection list and the steps in the edit JCL. So do not change the expanded JCL value after it has restarted using step restart.

Example 8-3 Sample JCL for Step Restart

```
//TWSSTEP  JOB (290400),'RESTART STEP',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H
//STEP1       EXEC PGM=MYPROG
//STEP2       EXEC PGM=IEFBR14
//STEP3       EXEC PGM=IEFBR14
//STEP4       EXEC PGM=IEFBR15
//STEP5       EXEC PGM=IEFBR14
```

8. Now you can specify the restart range from the Step Restart Selection List panel, but first we need to get there. As seen in Figure 8-8 on page 194, we can enter RC on the command line that brings us here operation Restart and Cleanup (Figure 8-9 on page 196). We have a few selections here:
   - Step Restart: Restart the operation, allowing the selection of the steps to be included in the new run.
   - Job Restart: Restart the operation from the beginning.
   - Start Cleanup: Start only the cleanup of the specified operation. Operation is not restarted.
   - Start Cleanup with AR: Start only the cleanup of the specified operation according to AR restart step when available. Operation is not restarted.
   - Display Cleanup: Display the result of the cleanup actions.

9. From the selections in the panel as described above and shown in Figure 8-9, we choose option 1 for Step Restart. You can edit the JCL if you like before the Restart by changing the Edit JCL option from N to Y. Now you will notice a bit of a hesitation by Tivoli Workload Scheduler for z/OS as it retrieves the joblog before proceeding.

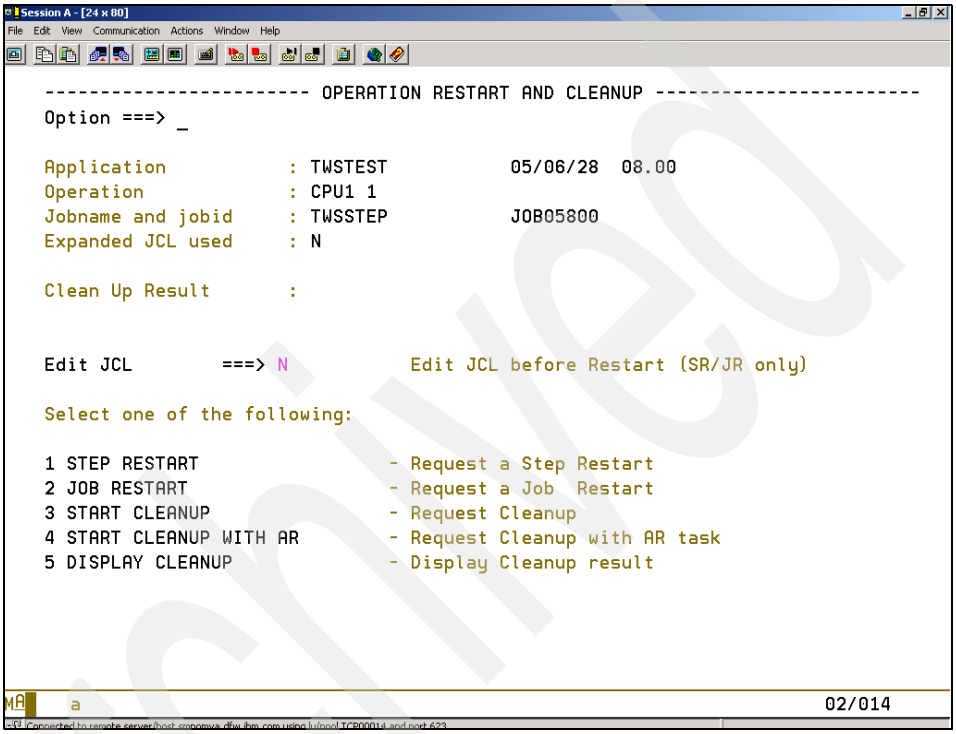


Figure 8-9 Operation Restart and Cleanup

10. Now we are at the Step Restart Selection List (Figure 8-10 on page 197). Here it shows that Step 1 is selected as the restart point. Here we use the S row command and then restart the job. The restarted job will end at step 5. (The end step defaults to the last step). You can also set the step completion code with a desired value for specific restart scenarios, using row command F for a step that is not executable. After entering S, we enter GO to confirm our selection, then exit using F3. You will also find a confirmation panel just as in a Simple Job Restart, the same thing applies here as it did in Figure 8-7 on page 193.

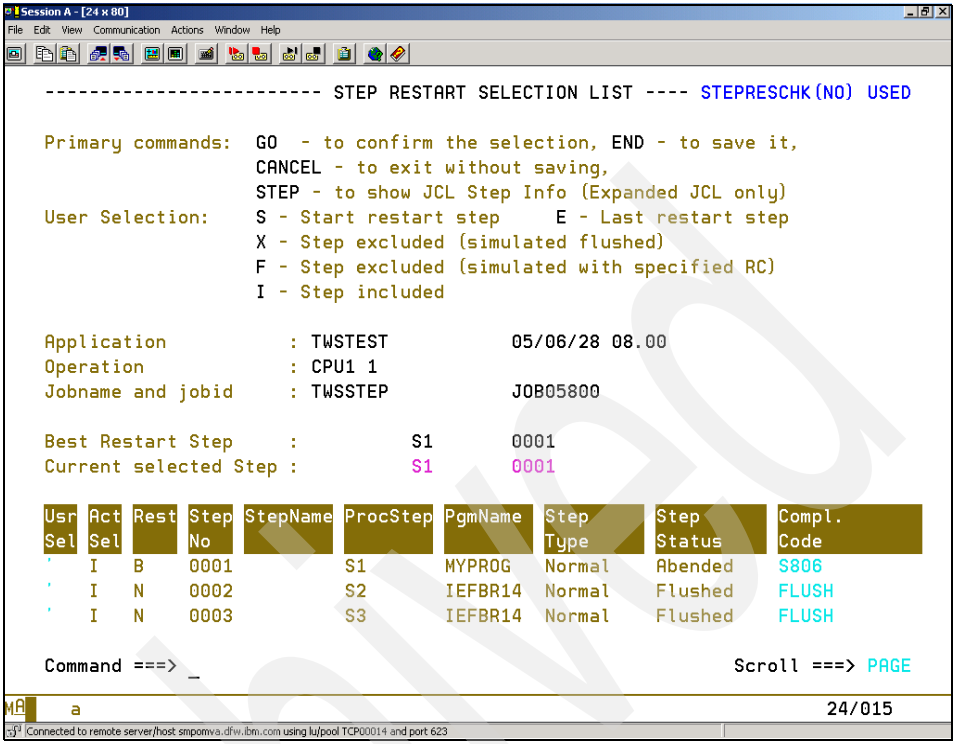


Figure 8-10 Step Restart Selection List

8.4 Steps that are not restartable

The cataloging, re-cataloging, and un-cataloging operations cannot by themselves hinder the capability to restart, because you can use EQQCLEAN. However, there are some cases where a step is not restartable:

- The step follows the abended step.
- The step includes a DDNAME that is listed in the parameter DDNOREST (in the RCLOPTS initialization statement).
- The step includes a DDNAME that is listed in the parameter DDNEVER (in the RCLOPTS initialization statement). In this case, the preceding steps are also not restartable.
- The step uses generation data group (GDG) data sets:
  - With a disposition different than NEW
  - With a relative number greater than zero and expanded JCL is not being used

- The step is a cleanup step.
- The step is flushed or not run, and the step is not simulated. The only exception is when the step is the first flushed in the JCL, all the following steps are flushed, and the job did not abend.
- The data set is not available, and the disposition is different than NEW.
- The data set is available, but all of the following conditions exist:
  - The disposition type is OLD or SHR.
  - The normal disposition is different from UNCT.
  - The data set has the disposition NEW before this step (The data set is allocated by this JCL.)
  - The data set has been cataloged at the end of the previous run and a cat action is done in one of the steps that follow.
- The step refers to a data set with DISP=MOD.
- To restart the job from this step entails the execution of a step that cannot be re-executed.

8.4.1 Re-executing steps

The step can be re-executed if it does not refer to any data sets. If the step does refer to a data set, it can be re-executed if the data set meets the one of the three following conditions:

- The disposition type is NEW.
- The disposition type is MOD, and the data set is allocated before running the step.
- The disposition type is OLD or SHR, and the data set is either of the following:
  - Allocated before running the step.
  - Available and has one of the following characteristics:
    - The normal disposition is UNCATLG.
    - The data set is not allocated in the JCL before this step.
    - The data set is cataloged before running this step.
    - The data set has been cataloged at the end of the previous run and no catalog action is done in one of the steps that follow.

Tivoli Workload Scheduler for z/OS suggests the best restart step in the job based on how the job ended in the prior run. Table 8-1 on page 199 shows how Tivoli Workload Scheduler for z/OS selects the best restart step.

Table 8-1 How Tivoli Workload Scheduler for z/OS selects the best restart step

How the job ended	Best restartable step
The job terminated in error with an abend or JCL error for non-syntactical reasons.	Last restartable step
There was a sequence of consecutive, flushed steps.	Last restartable step
The last run was a standalone cleanup.	First restartable step
All other situations.	First restartable step

8.4.2 EQQDELDS

EQQDELDS is a Tivoli Workload Scheduler for z/OS supplied program that deletes data sets based on the disposition specified in the JCL and the current status of the catalog. You can use this program if you want to delete data sets that are cataloged by your applications.

The JCL to run EQQDELDS and more detailed information are located in the member EQQDELDI, which is in the SEQQSAMP library.

8.4.3 Deleting data sets

The EQQDELDI member in the SEQQSAMP library has the JCL you need to run the sample program EQQDELDS. You can use EQQDELDS to delete data sets based on the disposition indicated in the JCL and the current status of the data set in the catalog. It is important to note that EQQDELDS is not a function of Tivoli Workload Scheduler; this program is provided by Tivoli Workload Scheduler for z/OS development in order to help customers who require this function. It helps customers primarily who do not want to change their existing application JCL.

To run this program, modify the JCL statements in the sample (SEQQSAMP library) to meet your standards. EQQDELDS deletes any data set that has a disposition (NEW,CATLG), zzz or (NEW, KEEP) for SMS, if the data set is already present in the catalog. It optionally handles passed data sets. It is important to note that data sets are not deleted if they are referenced in prior steps with DISP different from NEW.

EQQDELDS can be used to avoid `not catlgd2` error situations. EQQDELDS cannot run concurrently with subsequent steps of the job in which it is inserted. So if Smartbatch is active, you can define EQQDELDS with ACTION=BYPASS in the Smartbatch user control facility.

EQQDELDS supports the following types of delete processing:

- DASD data sets on primary volumes are deleted using IDCAMS.

- Tape data sets are deleted using IDCAMS NOSCRATCH. This does not cause mount requests for the specified tape volumes.
- DFHSM-migrated data sets are deleted using the ARCHDEL (ARCGIVER) interface. Data sets are moved to primary volumes (recalled) before deletion.

EQQDELDS logs all actions performed in text lines written to the SYSPRINT DD. A non-zero return code from IDCAMS or ARCHDEL causes EQQDELDS to end.

8.4.4 Restart jobs run outside Tivoli Workload Scheduler for z/OS

On some instances you may have jobs that run outside of Tivoli Workload Scheduler for z/OS that require a restart. You may have already had such a process in place when you migrated to Tivoli Workload Scheduler. In any event you can do a restart on such jobs. This will require the insertion of an additional job step at the beginning of the job to clean up simple data sets during the initial run of a job, in order to avoid “not catlg2” errors.

In the case of a rerun, this require copying this step immediately *before* the restart step. In Figure 8-4, we show a job that executes three production steps after the EQQDELDS statement (STEP0020, STEP0030, and STEP0040).

Example 8-4 EQQDELDS sample

```
//DELDSAMP JOB (9999,9999,999,999),CLASS=0,
//  MSGCLASS=I,NOTIFY=&SYSUID
//STEP0010 EXEC EQQDELDS <--To scratch files to avoid not catlg2 on
initial run
//STEP0020 EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//ADDS1 DD DSN=DATASETA.TEST.CAT1,
//   UNIT=SYSDA,SPACE=(TRK,(1,1),DISP=(NEW,CATLG)
//STEP0030  EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//ADDS2 DD DSN=DATASETA.TEST.CAT2,
//   UNIT=SYSDA,SPACE=(TRK,(1,1),DISP=(NEW,CATLG)
//STEP0040  EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//ADDS3 DD DSN=DATASETA.TEST.CAT3,
//   UNIT=SYSDA,SPACE=(TRK,(1,1),DISP=(NEW,CATLG)
//*
```

Now to restart this job in STEP0030, we insert the following DD card *immediately before* STEP0030 (the step to be restarted):

```
//STEP003A EXEC EQQDELDS
```

Note: The actual step name can be different from the one shown here, but the EXEC EQQDELDS must be as shown. The step name must be unique within the job.

Then put the restart statement in the job card to start at the step that executes EQQDELDS:

```
RESTART=(STEP003A)
```

Example 8-5 EQQDELDS restart

```
//DELDSAMP JOB (9999,9999,999,999),CLASS=O,
//  MSGCLASS=I,NOTIFY=&SYSUID,RESTART=(STEP003A)
//STEP0010 EXEC EQQDELDS
//STEP0020 EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//ADDS1 DD DSN=DATASETA.TEST.CAT1,
//   UNIT=SYSDA,SPACE=(TRK,(1,1),DISP=(NEW,CATLG)
//STEP003A EXEC EQQDELDS
//STEP0030  EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//ADDS2 DD DSN=DATASETA.TEST.CAT2,
//   UNIT=SYSDA,SPACE=(TRK,(1,1),DISP=(NEW,CATLG)
//STEP0040  EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//ADDS3 DD DSN=DATASETA.TEST.CAT3,
//   UNIT=SYSDA,SPACE=(TRK,(1,1),DISP=(NEW,CATLG)
//*
```

This executes the EQQDELDS process (to clean up data sets for this restart of the job) and then begin restarting the job at STEP0030.

9

Dataset triggering and the Event Trigger Tracking

This chapter provides information about setting up the ETT (Event Trigger Tracking) table and dataset triggering. The ETT tracks and schedules work based on jobs that run outside of Tivoli Workload Scheduler for z/OS Dataset Triggering is used whenever a data set with the same name as a Special Resource is created or read. In this chapter we go into detail about how to set up both the ETT and dataset triggering using Special Resources.

This chapter covers the following topics[2]:

- Dataset triggering
- Event Trigger Tracking

[2] Some of the content in this chapter was provided by Cy Atkinson, Anna Dason, and Art Eisenhower.

9.1 Dataset triggering

Dataset triggering in Tivoli Workload Scheduler for z/OS is a unique way in which the Tivoli Workload Scheduler for z/OS Tracker issues an SRSTAT. The SRSTAT command as described in *IBM Tivoli Workload Scheduler for z/OS Managing the Workload Version 8.2*, SC32-1263, enables you to change the overriding (global) availability, quantity, and deviation of a Special Resource. You can do this to prevent operations from allocating a particular resource or to request the ETT function to add an application occurrence to the current plan.

9.1.1 Special Resources

Special Resources can be set up to be any type of limited resource, such as a tape drive, communication line, or even a database. Creating a Special Resource is done through the panel (by entering `=1.6`) in Tivoli Workload Scheduler for z/OS (see Figure 9-1 on page 205). The Special Resource panel updates the resource database and uses the following details for each resource:

- Name: Up to 44 characters. This identifies the resource.
- Availability: Yes (Y) or No (N).
- Connected workstations: A list of the workstations where operations can allocate the resource.
- Quantity: 1 to 999999.
- Used for: How Tivoli Workload Scheduler for z/OS is to use the resource: for planning (P), control (C), both (B), or neither (N).
- On-error action: Free all (F), free exclusively-held resources (FX), free shared resources (FS), and keep all (K). Tivoli Workload Scheduler for z/OS uses the attribute specified at operation level first. If this is blank, it uses the attribute specified in the resource database. If this is also blank, it uses the ONERROR keyword of the RESOPTS statement.

The quantity, availability, and list of workstations could vary with time. To control a Special Resource you can create your own time intervals. You can also state whether Special Resource is shared or exclusive, as well as the quantity and on-error attributes.

Figure 9-1 shows the Maintaining Special Resources panel. From here you can select 1 to Browse the Special Resources, 2 to Create a Special Resource, and 3 to list Special Resources for browsing, modifying, copying, deleting, or creating.

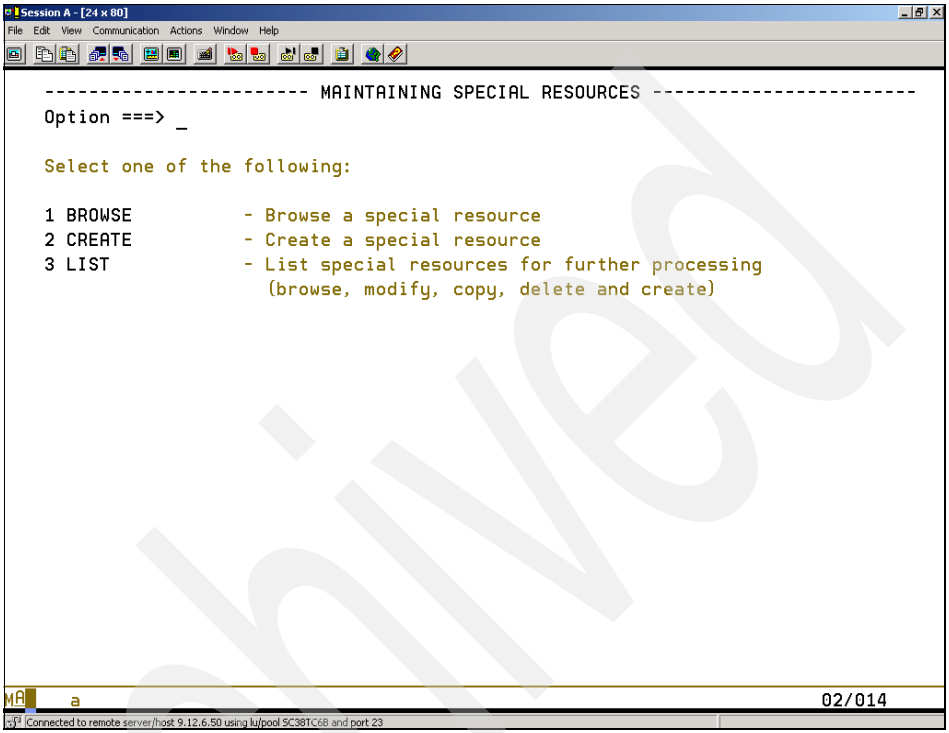


Figure 9-1 Special Resource Panel from Option =1.6

Figure 9-2 shows the list criteria when we chose option 3 (List). As the figure shows, you can list the Special Resource by name or choose a wild card format as indicated above. You can also list, based on the group ID, whether it uses Hiperbatch™ or not. In the TYPE OF MATCH field, you can specify whether it should be Exact, Prefix, or Suffix based on the * and % wild cards. Or you may leave it blank to be generic.

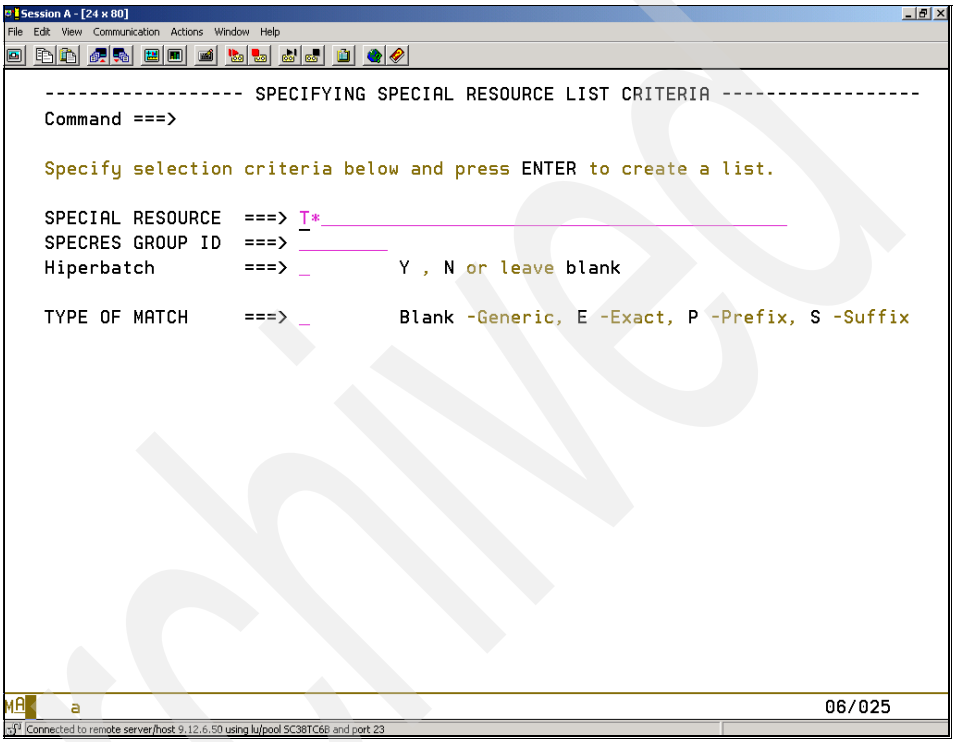


Figure 9-2 Special Resource List Criteria option 3

Figure 9-3 shows the results from using the wildcard format above. There is one command line option (CREATE).

- On the row commands, you can either Browse, Modify, Copy, or Delete the Special Resource.
- Following the Special Resource column, the name of the Special Resource (Specres Group ID) indicates what group the resource is a part of if any.
- A is for availability, which is indicated by a Y or an N. Thus, when an operation has a Special Resource it enters the current plan with the Special Resource's availability as stated.
- Qty is the quantity of the resource which can be in a range from 1 to 999999.
- Num Ivl is the number of intervals that exist for the resource.

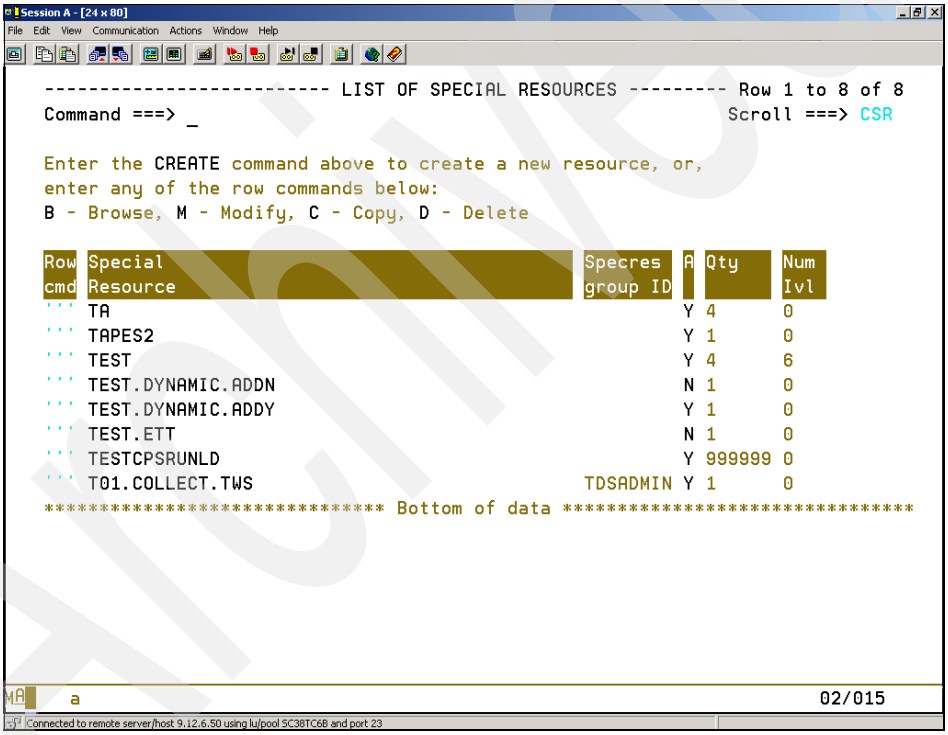


Figure 9-3 Special Resource List

9.1.2 Controlling jobs with Tivoli Workload Scheduler for z/OS Special Resources

When Tivoli Workload Scheduler for z/OS generates the workload for the day (creates the current plan), it searches the database for predecessors and successors. If a job has a predecessor defined that is *not* in the current plan for that day, Tivoli Workload Scheduler for z/OS does not add that predecessor to the job for that day. Because requested (ad hoc) or data set-triggered jobs do not show up in the plan until they arrive to execute, Tivoli Workload Scheduler for z/OS is not aware of when they will run. Thus, they are not included as predecessors in the current plan.

One solution to this is the use of Tivoli Workload Scheduler for z/OS Special Resources. Tivoli Workload Scheduler for z/OS Special Resources can be set to either Available = Yes or Available = No. The jobs executing the Special Resources functions have been defined with the last three characters of the job name being either AVY (for Available = Yes) or AVN (for Available = No). In Figure 9-4 on page 209, the scheduled job has a Special Resource requirement that will be made available when the requested/triggered (unscheduled) job runs.

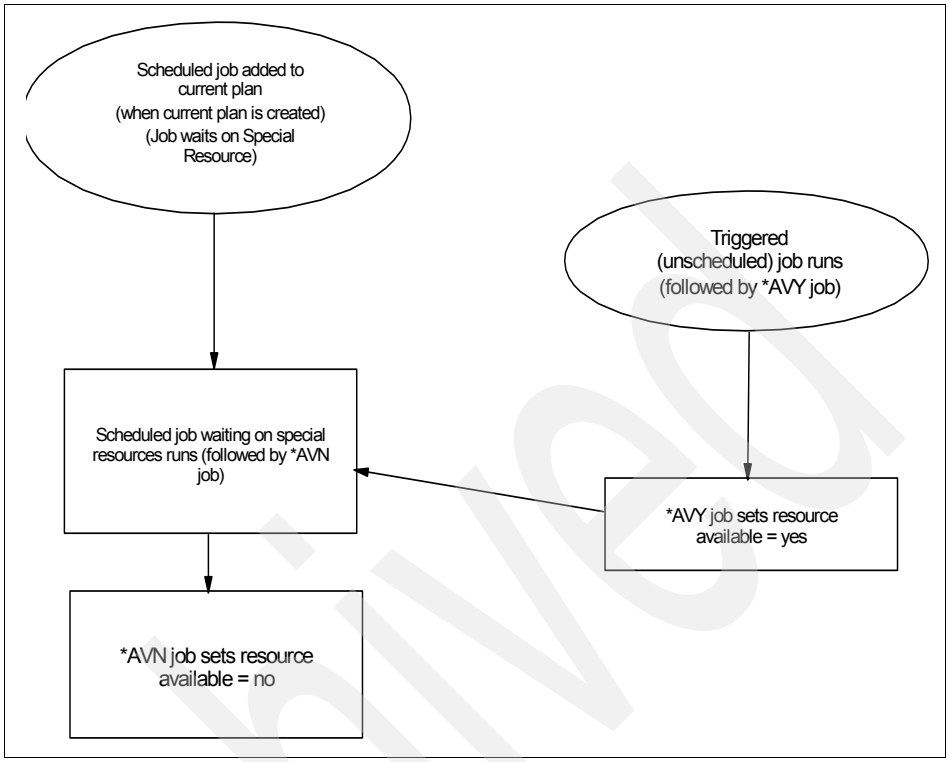


Figure 9-4 Using Tivoli Workload Scheduler for z/OS Special Resources

Where possible, the Special Resource names (in the form of a data set name) were designed to reflect the predecessor and successor job name. This may not be done in all instances because some predecessors could have more than one successor. The list of Special Resources can be viewed in Tivoli Workload Scheduler for z/OS under option 1.6 (see Figure 9-1 on page 205). To view which Special Resource a job is waiting on, browse the application in the database (option 1.4.3) and select the operation on the row command (S.3) as shown in Figure 9-5 on page 210.

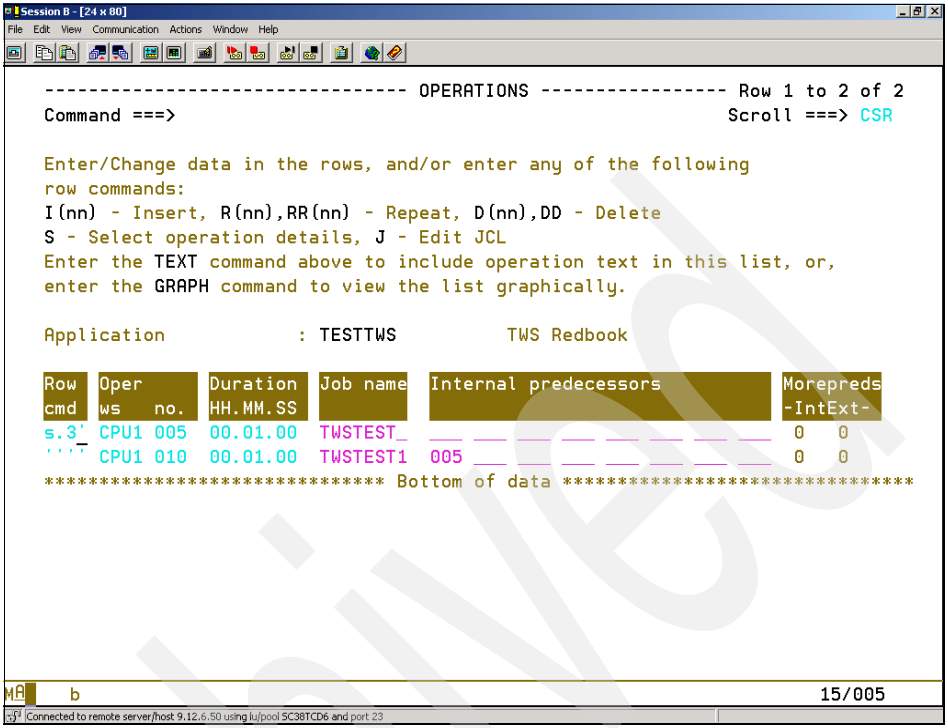


Figure 9-5 Row command S.3 for Special Resources for an operation

This displays the Special Resource defined to this job (Figure 9-6). The Special Resource jobs themselves can be put into a special joblib or any joblib of your choosing.

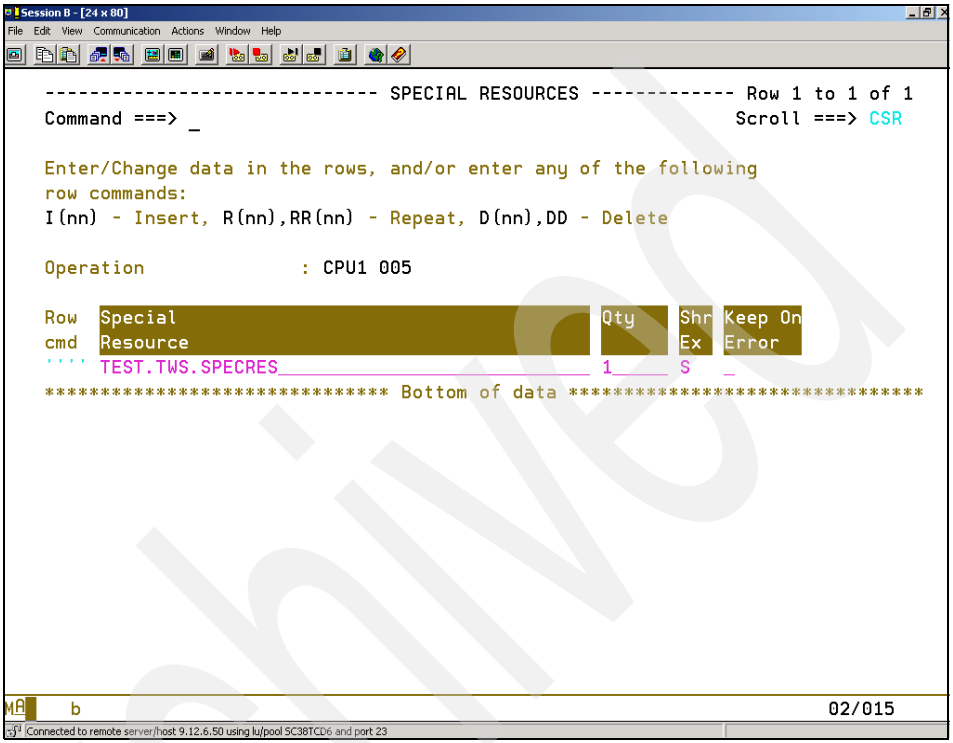


Figure 9-6 Sample Special Resource for an operation

9.1.3 Special Resource Monitor

You can use the Resource Object Data Manager (RODM) to track the status of real resources used by Tivoli Workload Scheduler for z/OS operations. RODM is a data cache that contains information about real resources at your installation. Products such as AOC/MVS report actual resource status to RODM; RODM reflects the status by updating values of fields in classes or objects that represent the real resources. Subsystems on the same z/OS image as RODM can subscribe to RODM fields. When RODM updates a field, all subscribers to the field are notified.

Tivoli Workload Scheduler for z/OS support for RODM lets you subscribe to RODM fields for fields in Special Resources. When RODM notifies a change,

Tivoli Workload Scheduler for z/OS updates resource fields that have a subscription to RODM. You can subscribe to RODM for these fields:

AVAILABLE The Available field in the resource. This value overrides the default and interval values.

QUANTITY The Quantity field in the resource. This value overrides the default interval values.

DEVIATION The Deviation field. Use this field to make a temporary adjustment to quantity. Tivoli Workload Scheduler for z/OS adds quantity and deviation together to decide the amount that operations can allocate. For example, if quantity is 10 and deviation is -3, operations can allocate up to 7 of the resource.

Specify these keywords to invoke monitoring through RODM:

RODMTASK Specified on the OPCOPTS statement for the Controller and for each Tracker that communicates with a RODM subsystem.

RODMPARM Specified on the OPCOPTS statement for the Controller and identifies the member of the parameter library that contains RODMOPTS statements.

RODMOPTS Specified for a Controller and contains destination and subscription information. A RODMOPTS statement is required for each field in every resource that you want to monitor. Each statement is used to subscribe to a field in an RODM class or RODM object for a field in a Special Resource. The RODM field value is used to set the value of the resource field. RODMOPTS statements are read when the Controller is started. When a Tracker that communicates with RODM is started, it requests parameters from the Controller. The Controller sends subscription information to the Tracker, which then subscribes to RODM. An event is created when RODM returns a value, which is used to update the Special Resource field in the current plan. Tivoli Workload Scheduler for z/OS does not schedule operations that use a Special Resource until RODM has returned the current field value and Tivoli Workload Scheduler for z/OS has updated the resource.

To use RODM monitoring, you must ensure that:

- A Tracker is started on the same z/OS image as the RODM subsystem that requests are sent to, and RODMTASK(YES) is specified for both the Tracker and the Controller.
- An Event Writer is started in the Tivoli Workload Scheduler for z/OS address space that communicates with RODM. This address space creates resource

events (type S) from RODM notifications, which Tivoli Workload Scheduler for z/OS uses to update the current plan.

- The Controller is connected to the Tracker through XCF, NCF, or a submit/release data set.
- Each address space has a unique RACF user ID if more than one Tivoli Workload Scheduler for z/OS address space communicates with an RODM subsystem, such as when you start production and test systems that subscribe to the same RODM subsystem.

Tivoli Workload Scheduler for z/OS does not load or maintain data models in the RODM cache, or require a specific data model. You need not write programs or methods to use RODM through Tivoli Workload Scheduler for z/OS or define specific objects or fields in RODM. Tivoli Workload Scheduler for z/OS does not update RODM-defined data.

RODM fields have several subfields. The RODM field that Tivoli Workload Scheduler for z/OS subscribes to must have a notify subfield. Through a subscription to this subfield, RODM notifies Tivoli Workload Scheduler for z/OS of changes to the value subfield. Tivoli Workload Scheduler for z/OS uses changes to the value subfield to monitor Special Resources. But only these data types are valid for Tivoli Workload Scheduler for z/OS RODM support:

Table 9-1 Valid RODM data types for value subfields

Abstract data type	Data type ID
CharVar(Char)	4
Integer (Bin 31)	10
Smallint (Bin 15)	21

Tivoli Workload Scheduler for z/OS maintains RODM status for all Special Resources in the current plan. You can check the current status in the Special Resource Monitor dialog. Each Special Resource has one of these values:

N (not monitored) The Special Resource is not monitored through RODM.

I (inactive) Monitoring is not currently active. Tivoli Workload Scheduler for z/OS sets this status for all subscriptions to an RODM subsystem that the Controller cannot communicate with. This can occur when communication is lost with RODM or with the Tracker. The Controller sets the value of each monitored field according to the RODMLOST keyword of RODMOPTS.

P (pending) Tivoli Workload Scheduler for z/OS has sent a subscription request to RODM, but RODM has not returned a value.

A (active) Tivoli Workload Scheduler for z/OS has received a value from RODM and the Special Resource field has been updated.

The names of RODM classes, objects, and fields are case-sensitive. Ensure you preserve the case when specifying RODMOPTS statements in the parameter library. Also, if a name contains anything other than alphanumeric or national characters, you must enclose the name in quotation marks.

If Tivoli Workload Scheduler for z/OS subscribes to RODM for a resource that does not exist in the current plan and the DYNAMICADD keyword of RESOPTS has the value YES or EVENT, the event created from the data returned by RODM causes a dynamic add of the resource. DYNAMICADD is described further in 9.1.5, “DYNAMICADD and DYNAMICDEL” on page 219.

If a request from Tivoli Workload Scheduler for z/OS cannot be processed immediately because, for example, long-running programs in RODM access the same data that Tivoli Workload Scheduler for z/OS requests need access to, be aware of possible delays to operation start times.

You can access the Special Resource Monitor in the Tivoli Workload Scheduler for z/OS dialog by entering option =5.7. See Figure 9-7.

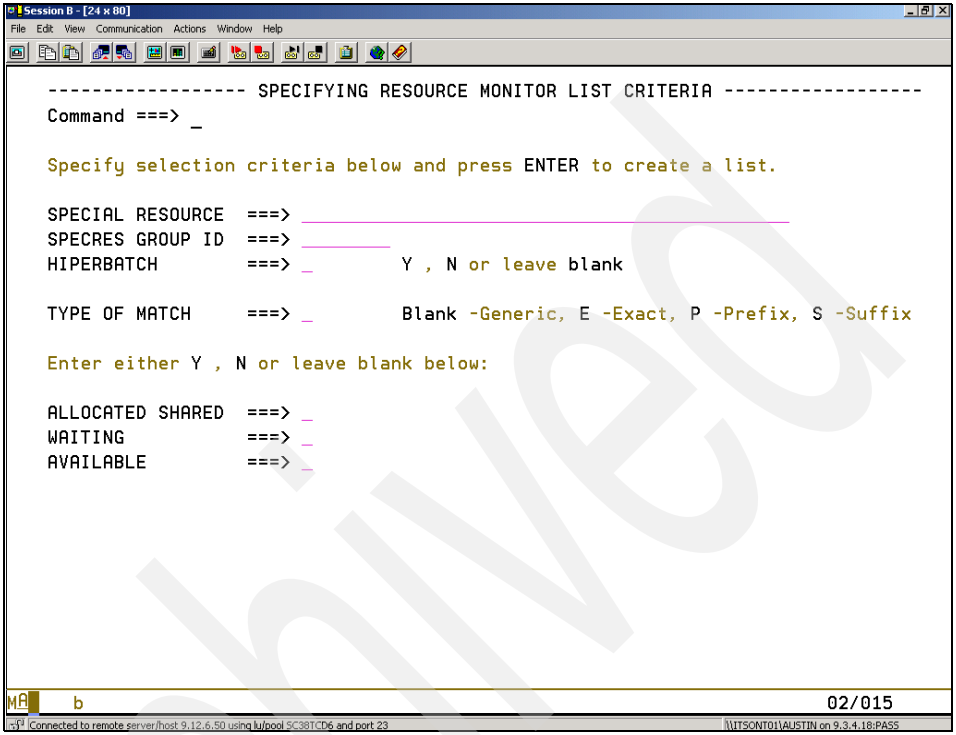


Figure 9-7 Specify Resource Monitor List Criteria (option =5.7)

Figure 9-7 looks similar to the Special Resource List Criteria panel (Figure 9-2 on page 206). The difference is the Allocated Shared, Waiting, and Available options, which can be left blank or selected with Y or N:

- Allocated Shared (all selections are optional):
  - **Y** Selects only resources allocated shared.
  - **N** Selects only resources allocated exclusively.
  - Left blank Selects both allocation types.
- Waiting (all selections are optional):
  - **Y** Selects only resources that operations are waiting for.
  - **N** Selects only resources that no operations are waiting for.
  - Left blank Includes all resources.
- Available (all selections are optional):
  - **Y** Selects only resources that are available.
  - **N** Selects only resources that are unavailable.
  - Left blank Includes all resources.

Figure 9-8 shows the Special Resource Monitor display. The row commands are for Browse, Modify, In use list, and Waiting queue.

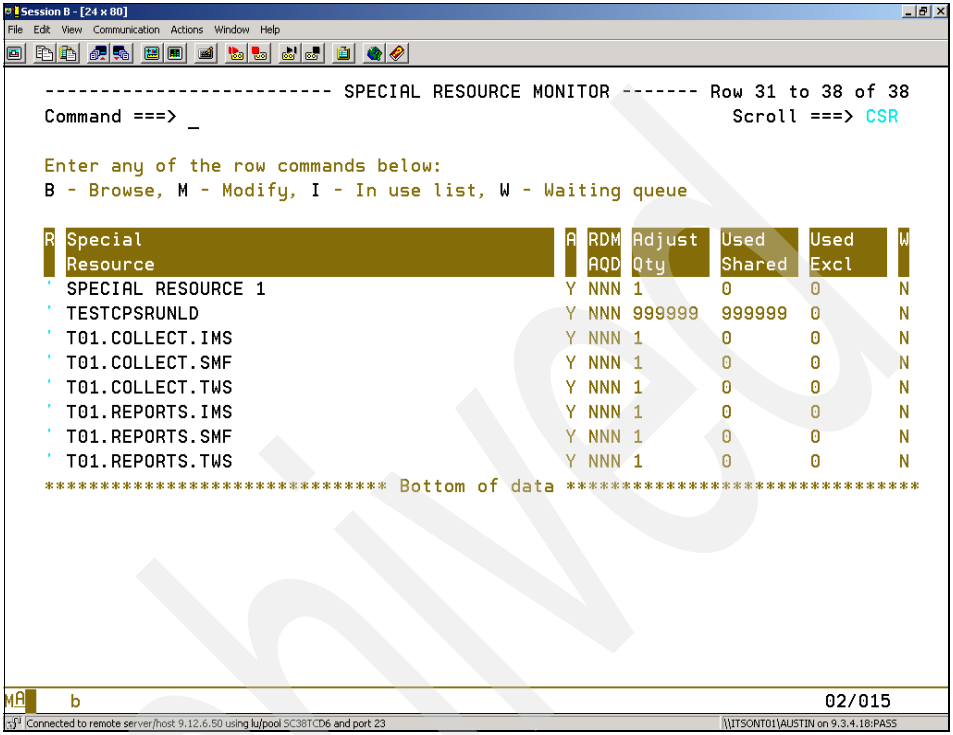


Figure 9-8 Special Resource Monitor

Figure 9-9 shows the Modifying a Special Resource panel.

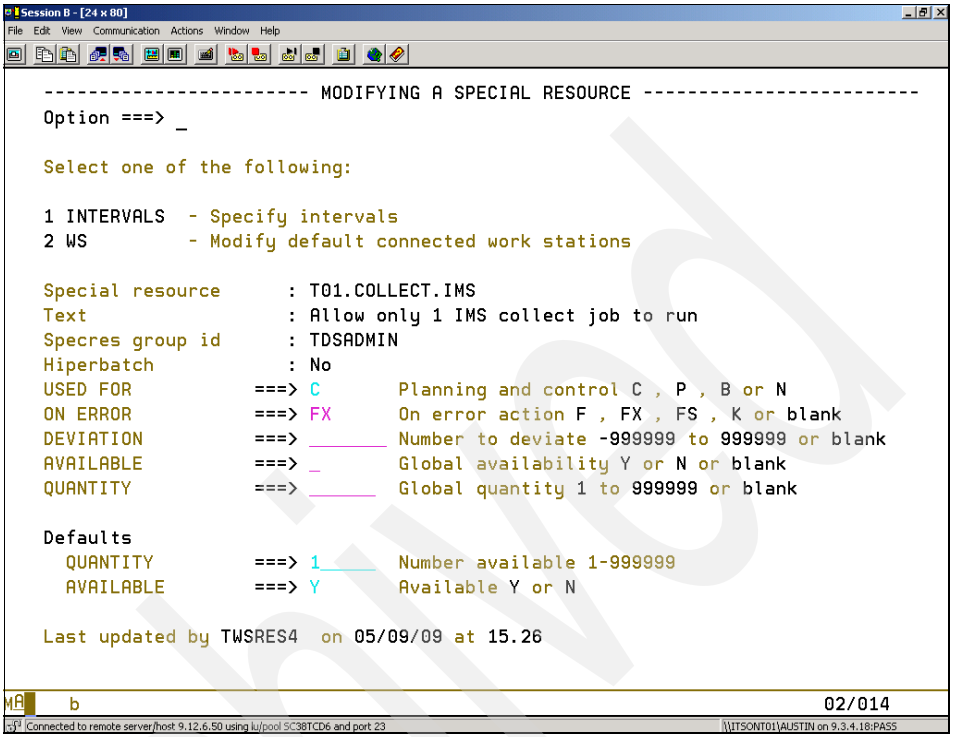


Figure 9-9 Modifying a Special Resource

This panel is exactly the same as the Browse panel except for that you can modify the Special Resource (in browse you cannot). The fields are:

- Special Resource: The name of the resource.
- Text: Description of the resource.
- Specres group ID: The name of a group that the resource belongs to. You can use group IDs as selection criteria.
- Last updated by: Shows the user ID, date, and time the resource was last updated. If the user ID field contains *DYNADD* the resource was dynamically added at daily planning. Or if it contains *SUBMIT* the resource was added when an operation was submitted.
- Hiperbatch (required; specify whether the resource is eligible for Hiperbatch):

  Y The resource is a data set eligible for Hiperbatch.

  N The resource is *not* eligible for Hiperbatch.

- USED FOR (required; specify whether the resource is used for planning and control functions):

  P Planning

  C Control

  B Both planning and control

  N Neither planning nor control

- ON ERROR (optional; specify the action taken when an operation using the resource ends in error):

  F Free the resource.

  FS Free if allocated shared.

  FX Free if allocated exclusively.

  K Keep the resource.

  Blank Use the value specified in the operation details. If this value is also blank, use the value of the ONERROR keyword on the RESOPTS initialization statement.

  The action is taken only for the quantity of the resource allocated by the failing operation.

- DEVIATION (optional): Specify an amount, -999999 to 999999, to be added to (positive number) or subtracted from (negative number) the current quantity.

  Deviation is added to the current quantity value to determine the quantity available. For example, if quantity is 5 and deviation is -2, operations can allocate up to 3 of the resource.

- AVAILABLE (optional): Specify whether the resource is currently available (Y) or not available (N). This value overrides interval and default values.

- QUANTITY (optional): Specify a quantity, 1 to 999999. This value overrides interval and default values.

- Defaults: Specify values that are defaults for the resource. Defaults are used if no value is specified at interval level or in a global field.

  – QUANTITY (required): Specify a quantity, 1-999999.

  – AVAILABLE (required): Specify whether the resource is available (Y) or not available (N).

9.1.4 Special Resource Monitor Cleanup

To get an obsolete Special Resource out of the Special Resource Monitor panel:

- If the resource is defined in the RD database, it will be removed automatically from the Special Resource Monitor by the Current Plan Extend or Replan processing when the following conditions are met:
  a. There is no operation remaining in the current plan that uses that Special Resource.
  b. The R/S Deviation, Global Availability, and Global Quantity are all blank. These values can be blanked out manually via the dialogs, or an SRSTAT can be issued with the Reset parameter as documented in *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265.
- If the superfluous resources are not defined in the RD database, they have been created by DYNAMICADD processing, and to remove them, you must run a Current Plan Extend or Replan batch job with OPCOPTS option DYNAMICDEL (YES).

9.1.5 DYNAMICADD and DYNAMICDEL

DYNAMICADD {YES|NO} determines whether Tivoli Workload Scheduler for z/OS creates a Special Resource during planning if an operation needs a resource that is not defined in the Special Resource database.

Specify YES if you want Tivoli Workload Scheduler for z/OS to create a resource in the current plan. The Special Resource database is not updated.

Specify NO if Tivoli Workload Scheduler for z/OS should not dynamically create a resource. Tivoli Workload Scheduler for z/OS plans the operation as if it does not use the resource.

A dynamically created resource has these values:

- Special Resource: The name specified by the allocating operation.
- Text: Blank.
- Specres group ID: Blank.
- Hiperbatch: No.
- Used for: Both planning and control.
- On error: Blank. If an error occurs, Tivoli Workload Scheduler for z/OS uses the value specified in the operation details or, if this field is blank, the value of the ONERROR keyword of RESOPTS.

- Default values: The resource has these default values for quantity and availability:
  - Quantity: The amount specified in the first allocating operation. The quantity is increased if more operations plan to allocate the Special Resource at the same time. Tivoli Workload Scheduler for z/OS increases the quantity only for dynamically created resources to avoid contention.
  - Available: Yes.
  - Intervals: No intervals are created. The default values specify the quantity and availability.
  - Workstations: The resource has default value *, which means all workstations. Operations on all workstations can allocate the resource.
- The DYNAMICADD keyword of RESOPTS controls the dynamic creation of undefined Special Resources in the current plan.

DYNAMICDEL {YES|NO}: This parameter determines whether a Special Resource that has been dynamically added to the current plan can be deleted if the current plan is changed, without checking the normal conditions listed in the "Setting the global values" section of *IBM Tivoli Workload Scheduler for z/OS Managing the Workload Version 8.2*, SC32-1263. Specify NO if dynamically added changes can be deleted when the current plan is changed without further checking. Specify YES if dynamically added changes can be deleted when the current plan is changed.

9.1.6 RESOPTS

The RESOPTS statement defines Special Resource options that the Controller uses to process ready operations and Special Resource events. RESOPTS is defined in the member of the EQQPARM library as specified by the PARM parameter in the JCL EXEC statement (Figure 9-10 on page 221).

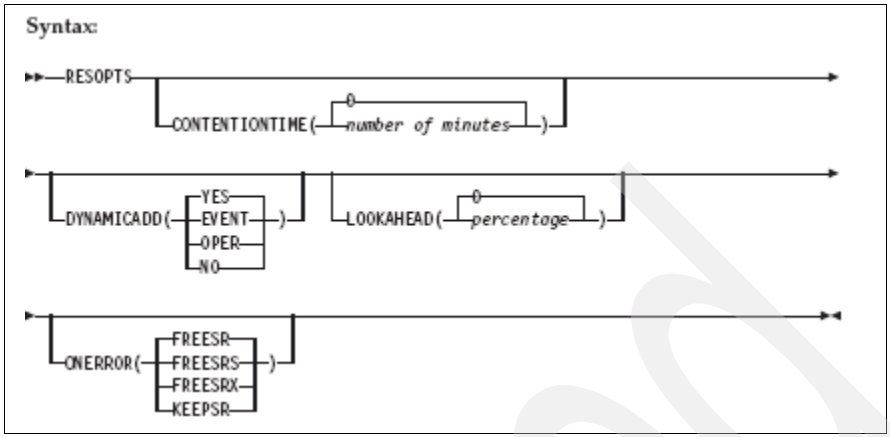


Figure 9-10 RESOPTS syntax

- CONTENTIONTIME: This parameter determines how long an operation remains on the waiting queue for a Special Resource before Tivoli Workload Scheduler for z/OS issues message EQQQ515W. Specify a number of minutes (1 to 9999) that an operation must wait before Tivoli Workload Scheduler for z/OS issues message EQQQ515W. When issued, the message is not repeated for the same Special Resource and operation, although Tivoli Workload Scheduler for z/OS can issue more than one message for an operation if it is on more than one waiting queue. You also must specify an alert action for resource contention on the ALERTS statement or the message will not be issued.
- DYNAMICADD(EVENT|OPER|NO|YES): If a Special Resource is not defined in the current-plan extension file or Special Resource database, DYNAMICADD determines whether Tivoli Workload Scheduler for z/OS creates a Special Resource in response to an allocate request from a ready operation or to a resource event created through the EQQUSIN or EQQUSINS subroutine, SRSTAT TSO command, API CREATE request, or a RODM notification.
  - Specify YES, which is the default value, if Tivoli Workload Scheduler for z/OS should create a Special Resource in the current plan. Tivoli Workload Scheduler for z/OS uses defaults to create the resource if the Special Resource database is not updated. When creating the resource, Tivoli Workload Scheduler for z/OS selects field values in this order:
    i. Values supplied by the allocating operation or event. An operation can specify a quantity; an event can specify quantity, availability, and deviation.
    ii. Tivoli Workload Scheduler for z/OS defaults.

– Specify NO if Tivoli Workload Scheduler for z/OS should not dynamically create a Special Resource. If an operation attempts to allocate the Special Resource, it receives an allocation failure, and the operation remains in status A or R with the extended status of X. If a resource event is received for the undefined resource, an error message is written to the Controller message log.
– Specify EVENT if Tivoli Workload Scheduler for z/OS should create a Special Resource in the current plan, only in response to a resource event. Resources are not created by operation allocations. But if the CREATE keyword of an SRSTAT command has the value NO, the Special Resource is not created.
– Specify OPER if Tivoli Workload Scheduler for z/OS should create a Special Resource in the current plan, only in response to an allocate request from a ready operation. Resources are not created by events. A dynamically created resource has the values shown in Figure 9-11 on page 223 if no description is found in the database.

Special resource	The name specified by the allocating operation or resource event.
Text	Blank.
Specres group ID	Blank.
Hiperbatch	No.
Used for	Control.
On error	Blank. If an error occurs, IBM Tivoli Workload Scheduler for z/OS uses the value specified in the operation details or, if this field is also blank, the value of the ONERROR keyword of RESOPTS.
Available	The value specified by an event (Y or N) or blank.
Quantity	The value specified by an event (1 to 999999) or blank.
Deviation	The value specified by an event (-999999 to 999999) or blank.
Default values	The resource has these values that are defaults for quantity and availability: **Quantity** 1. Or the quantity specified by an allocating operation. The default quantity is automatically increased if contention occurs, but only for dynamically created resources. **Available** Yes.
Intervals	No intervals are created.
Workstations	The resource has default value *, which means all workstations. Operations on all workstation can allocate the resource.

Figure 9-11 Dynamically created resource fields

Also see the DYNAMICADD keyword of BATCHOPT (Example 5-20 on page 139), which controls the dynamic creation of undefined Special Resources during planning. If Tivoli Workload Scheduler for z/OS subscribes to an RODM class or object for a resource that does not exist in the current plan, the event created from the data returned by RODM causes a dynamic add of the resource, if DYNAMICADD has the value YES or EVENT.

- LOOKAHEAD(percentage|0): Specify this keyword if you want Tivoli Workload Scheduler for z/OS to check before starting an operation whether there is enough time before the resource becomes unavailable. You specify the keyword as a percentage of the estimated duration. For example, if you do not want Tivoli Workload Scheduler for z/OS to start an operation unless the required Special Resource is available for the whole estimated duration, specify 100. Specify 50 if at least half the estimated duration must remain until the resource is due to be unavailable. If you specify LOOKAHEAD(0), which is

also the default, the operation is started if the Special Resource is available, even if it will soon become unavailable. Tivoli Workload Scheduler for z/OS uses this keyword only if the Special Resource is used for control.

- ONERROR(FREESRS|FREESRX|KEEPSR|FREESR): This keyword defines how Special Resources are handled when an operation using Special Resources is set to ended-in-error status. The value of the ONERROR keyword is used by Tivoli Workload Scheduler for z/OS only if the ONERROR field of a Special Resource in the current plan is blank and the Keep On Error value in the operation details is also blank.

  You can specify these values:

  – FREESR: Tivoli Workload Scheduler for z/OS frees all Special Resources allocated by the operation.

  – FREESRS: Tivoli Workload Scheduler for z/OS frees shared Special Resources and retains exclusively allocated Special Resources.

  – FREESRX: Tivoli Workload Scheduler for z/OS frees exclusively allocated Special Resources and retains shared Special Resources.

  – KEEPSR: No Special Resources allocated by the operation are freed. Tivoli Workload Scheduler for z/OS frees or retains only the quantity allocated by the failing operation. Other operations can allocate a Special Resource if the required quantity is available. Special Resources retained when an operation ends in error are not freed until the operation gets status complete.

    You can specify exceptions for individual resources in the Special Resources database and in the current plan.

Figure 9-12 shows a sample RESOPTS statement.

```
RESOPTS CONTENTIONTIME(10)      1
        DYNAMICADD(YES)         2
        ONERROR(FREESRS)        3
        LOOKAHEAD(200)          4
```

Figure 9-12 RESOPTS example

In this example:

1. Tivoli Workload Scheduler for z/OS issues message EQQQ515W if an operation has waited 10 minutes to allocate a Special Resource.
2. If a Special Resource is not defined in the current plan, Tivoli Workload Scheduler for z/OS creates the Special Resource in response to an allocate request from a ready operation or to a Special Resource event.

3. Shared Special Resources are freed if the allocating operation ends in error. Exclusively allocated Special Resources are kept.
4. If there is less than twice (200%) an operation's estimated duration left before the resource is due to become unavailable, Tivoli Workload Scheduler for z/OS will not start the operation.

9.1.7 Setting up dataset triggering

Dataset triggering is a Tivoli Workload Scheduler for z/OS Tracker function that monitors SMF14, SMF15, and SMF64 records via the Tracker subsystem and code in the IEFU84 SMF exit. It creates a Special Resource Status Change Even when there is a match on a Dataset Name in an SMF record and an entry in the Tracker's EQQDSLST.

There are two types of dataset triggers:

- The creation of a data set resource
- Setting the resource to available

Setting up dataset triggering with ETT:

1. Dataset triggering works by intercepting and examining all SMF data set CLOSE records.
2. You must compile and install the provided IEFU83 job-tracking exit. If you wish to trigger only when a data set is closed after creation or update, and when it is closed after an open for read (input), set the SRREAD parameter in the EQQEXIT macro in EQQU831 to SRREAD=NO.
3. Create a series of EQQLSENT macros as described in *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264, and from the example in Figure 9-13 on page 226, assemble and create an EQQDSLST and place it in the PDS specified by the EQQJCLIB DD statement in the Tracker start process.
4. Define a Special Resource ETT trigger in the Tivoli Workload Scheduler for z/OS Controller panel, specifying the data set names from the EQQLSENT macros as the Special Resource names:
   - SRREAD={Yes|NO|NONE}: An optional keyword defining whether a resource availability event should be generated when a data set is closed after being opened for read processing. When YES is specified or defaulted, an SR event is generated each time a data set is closed after being opened for either read or output processing. When you specify NO, the SR event is generated only when a data set has been opened for output processing. The event is not generated if the data set has been opened for read processing.

– USERID=: You need to have SETUID=YES coded in the IEFUJI SMF macro if you want to check by USERID in EQQDSLST.

Figure 9-13 shows an example of the EQQLSENT macro.

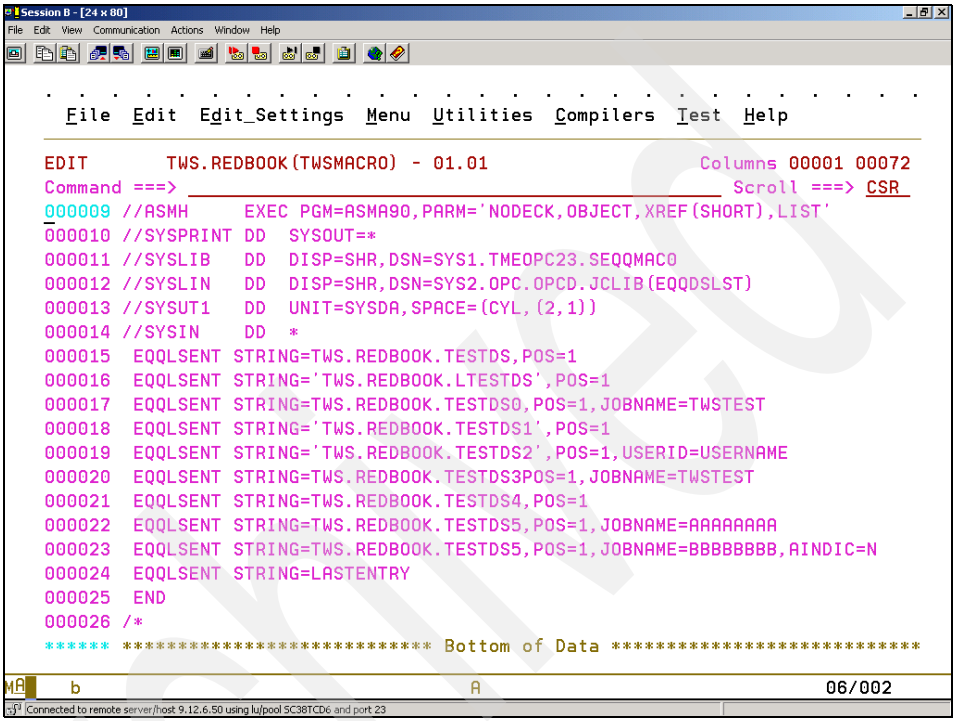


Figure 9-13 Sample EQQLSENT macro

In the first string in Figure 9-13 on line 15, when a string is not enclosed in single quotes and does not end with a blank, the data set name is a generic name or indicates it is a wild card. Enclosing the data set name with single quotes (line 16) works the same way if there is not a trailing blank. Line 18 is a generic request that will be triggered only when the job name TWSTEST creates it.

On line 19, this data set name string is for an absolute job name because the string ends with a blank and is enclosed in single quotes. This is a fully qualified data set name. On line 20 this data set name string is for an absolute name because the string ends with a blank and is enclosed in single quotes, and also is using user ID filtering. The data set name must be created by this user ID before triggering is performed. Line 21 the string is not enclosed in single quotes and does not end with a blank, making it a generic data set name but also has job name filtering.

On line 22, the string is not enclosed in single quotes and does not end with a blank; this makes it generic. In line 23, AINDIC={Y|N} is an optional keyword specifying whether the Special Resource should be set to available (Y) or unavailable (N). The default is to make the resource available. When job AAAAAAAA executes and creates TWS.REDBOOK.TESTDS5, it sets the resource available. After job BBBBBBBB updates the resource, it will set the resource to unavailable.

It is important to remember that whenever you update the macros, it is imperative that you:

1. Compile them using the EQQLSJCL member of the SEQQSAMP library.
2. Name the output file `EQQDSLST`.
3. Place it in the PDS specified by the “EQQJCLIB” DD statements in the Tracker start procedure.
4. Next, run this command for all Trackers from the ISPF log enter (SSNM is the Tracker started procedure name):

   ```
   /F SSNM,NEWDSLST
   ```

5. Check the MLOG to ensure that there were no problems; otherwise the triggers may not work. The command also works each time the Tracker is refreshed (for example, in an IPL, it will automatically attempt to load the EQQDSLST). It is essential to use the ISPF command above after each update and compile has been done.

After this is done, you can define the Special Resource ETT (see “Event Trigger Tracking” on page 228) triggers in the Tivoli Workload Scheduler for z/OS dialog by specifying the data set names you defined in the EQQLSENT macros as the Special Resources names.

After this all is completed, whenever any data set is closed, the IEFU83 exit is called on to examine the SMF close record. Then the exit will invoke the Tracker, which will compare the data set name and any other information from the SMF record against that of the EQQDSLST. So if there is a match, the Tracker then creates a Special Resource status change event record and puts it on the WTRQ in ECSA. (See more information in *IBM Tivoli Workload Scheduler for z/OS Diagnosis Guide and Reference Version 8.2*, SC32-1261.)

When browsing the Event data set, the SYY will appear in columns 21-23. The Tracker Event Writer task then moves the record from the WTRQ, writes it over to the Event data set, and sends it over to the Controller. The Controller then processes the event the same as an SRSTAT batch job would. Then the data set name is compared with those in the ETT and if there is a match, that application is added to the current plan.

9.1.8 GDG Dataset Triggering

When the Dataset Triggering code in the Tracker subsystem recognizes that the data set name in an SMF record is a GDG, it strips out the low-level qualifier from that data set name and matches only on the GDG base name. Also the Special Resource Status Change Event (SYY event) is created with only the GDG base name.

If the data set name is not recognized as a GDG, then the fully qualified data set name is used in the match and is included in any SYY event that may be created. The EQQLSENT macros that define the EQQDSLST can be coded to read only a portion of the data set name, thus allowing a basic level of generic matching.

To ensure that z/OS correctly recognizes all GDGs regardless of how they are created or manipulated, it is strongly recommended that all users code the GDGNONST(YES) option in their TRACKER OPCOPTS initialization statement:

```
OPCOPTS GDGNONST(YES)
```

On the Controller side, Special Resource ETT processing matches the Special Resource name in the incoming SYY event with the Triggering Event name in the ETT table. Generic Matching is allowed, but again it is strongly recommended that if the triggering data set is a GDG, the ETT definition should contain only the GDG base name.

Generic matching is unnecessary and incurs significant additional Controller overhead as compared to an exact match. Also, if a generic match is not coded correctly, there will be no match, and ETT.

9.2 Event Trigger Tracking

Event Trigger Tracking (ETT) is strictly a Controller function that detects certain events. It matches them against the definitions in the SI data set (created via dialog option 1.7). It then adds specified applications into the current plan. You can specify whether, when the new occurrence is added, any external dependencies are to be resolved.

The names of the ETT trigger events can be defined using wildcard characters to simplify your setup and reduce the number of definitions required. It is important to remember that while ETT matching of potential trigger events against the database for an exact match is extremely fast and efficient, searching the table for a generic match is much slower and less efficient. There is an initialization option (JTOPTS ETTGENSEARCH(YES/NO)) to enable you to completely turn off the generic search if you have no generically coded trigger names. Aside from

normal Job Tracking functionality, there is no Tracker involvement in ETT. The Tracker is wholly responsible for Dataset Triggering, but that is not ETT.

9.2.1 ETT: Job Trigger and Special Resource Trigger

ETT comes in two flavors: Job Trigger and Special Resource Trigger. Job Trigger processing is very simple:

1. When the Controller receives notice that any job or started task has entered the system, the jobname is matched first against the scheduled work in the current plan.
2. If there is a match, the job is tracked against the matching operation.
3. If there is no match against work already in the plan, the jobname is matched against the list of Job Triggers in the ETT definitions.
4. If there is a match on the ETT definitions, an occurrence of the associated application is added into the current plan.
5. If the JR (Jobname Replace or “Track the Trigger”) flag is set in the ETT definition, the jobname of the first operation in the application is changed to that of the Trigger Job, and the Trigger is tracked as the first operation.

Important: Any single job *either* will be tracked against an existing operation in the plan or will be an ETT Trigger. It cannot be both.

A Tivoli Workload Scheduler for z/OS scheduled job cannot be an ETT Trigger, but it can issue an SRSTAT and the SRSTAT can be a Special Resource ETT trigger. There is absolutely no security checking for Job Trigger ETT, and the Triggering job does not even have to run.

- You can submit a completely invalid JCL, and as long as there is a jobcard with a recognizable jobname, triggering will occur.
- You can define a Jobname Trigger as Test, then go to the console and type `START TEST` and Triggering will occur. It makes absolutely no difference whether there is a procedure named TEST anywhere in your proclibs.

9.2.2 ETT demo applications

The demo applications below are in two configurations. If we are doing Jobname Replace, there are two operations:

- The first is on a CPU workstation, and has the Submit option set to No, Hold/Release=Yes, and has an initial jobname of ETTDUMMY.
- The second is on a WTO workstation and simply writes a message to the console, then sets itself to C (completed) status.

All other demo applications have only the WTO operation. Figure 9-14 on page 231 shows a list of the demo applications for ETT Job Triggers. On the ETT table:

ET Shows the type of event that will be tracked for either a job or a Special Resource and will generate a dynamic update of the current plan by adding the associated application:

- **J** A job reader event is the triggering event.
- **R** A Special Resource is available and will perform a triggering event.

JR Shows job-name replace, which is valid with event type J only. It indicates whether the job name of the first operation in the associated application should be replaced:

- **Y** The name of the first operation is replaced by the job name of the triggering job.
- **N** The application is added unchanged.

When JR is a Y, the first job name in the application must be CPU# set up with submit off. This allows for external jobs to be tracked by Tivoli Workload Scheduler/OPC and may have other jobs dependent on the completion of this job or jobs in the flow. It may be necessary to track jobs submitted by a started tasks or MVS system submitted jobs such as SMF LOGS, SAR archive reports, IMS™ and or CICS® jobs.

DR Shows the dependency resolution: whether external dependencies should be resolved automatically when occurrences are added to the current plan or what should be resolved:

- **Y** External dependencies will be resolved.
- **N** External dependencies will NOT be resolved.
- **P** Only external predecessors will be resolved.
- **S** Only external successors will be resolved.

AS The Availability Status switch indicator. Only valid if the event type is R. Indicates whether ETT should add an occurrence only if there is a true availability status switch for a Special Resource from status available=no to available=yes, or if ETT should add an occurrence each time the availability status is set to available=yes, regardless of the previous status of the Special Resource.

For event type J this field must have the value N or blank. Y means that ETT adds an occurrence only when there is a true availability status switch from status available=no to available=yes, N means that ETT adds an occurrence each time the availability status is set to available=yes. If AS is set up with a Y, the ETT triggering will be performed only once and will not

be able to perform any addition triggering until the current plan runs and sets its status to unavailable.

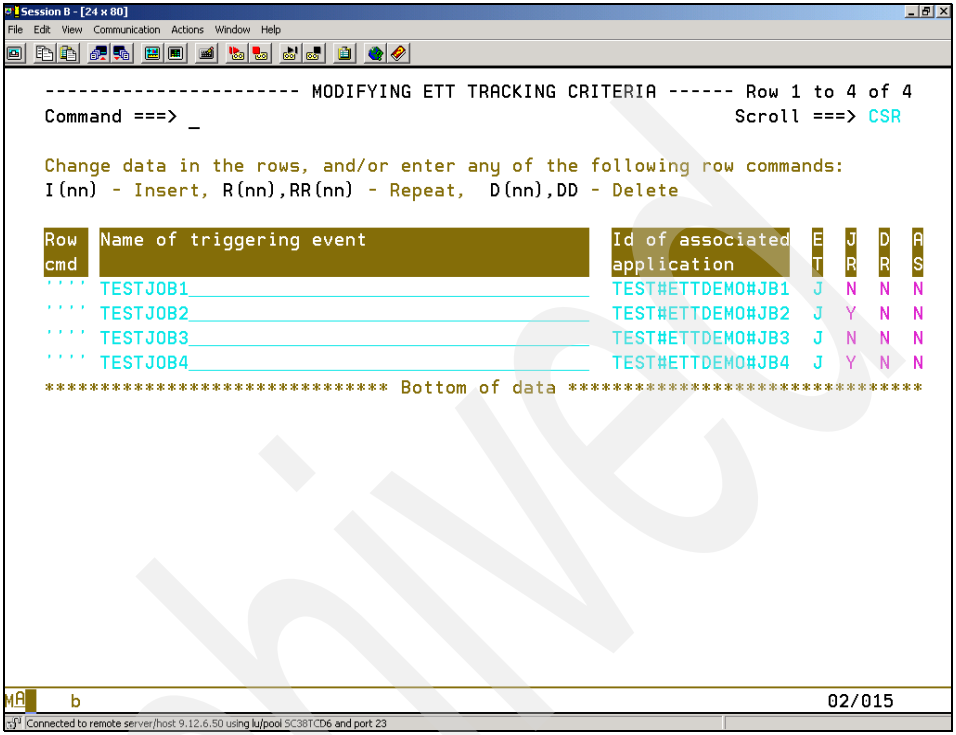


Figure 9-14 Demo applications for ETT Job Triggers

Trigger jobs 1 and 2 have a delay so we can see the difference between the two JR options.

1. When we submit TESTJOB1, TEST#ETTDEM#JB1 is added to the current plan and begins to run immediately. Because the JR flag is set to N, it does not wait for TESTJOB1 to complete.
2. When we submit TESTJOB2, TEST#ETTDEM#JB2 is added to the current plan but it waits for TESTJOB2 to complete before starting. So TESTJOB2 becomes the first operation of the occurrence and must be set to C (completed) status before anything else will happen. This is all because JR is set to Y.
3. If we go to the system console and issue the command START TESTJOB3, that command will fail because there is no such procedure in the proclib. TEST#ETTDEM#JB3 is still added to the current plan and will run immediately because JR is set to N.

4. If we submit TESTJOB4, it will fail do to security trying to allocate a data set with an HLQ that is locked out. This ETT definition has JR set to Y and now the job appears on the error queue, and the second operation in the application will not run.

9.2.3 Special Resource ETT

Special Resource is not a data set; it is an entry in a Tivoli Workload Scheduler for z/OS database that has a name that looks something like a data set name. There may be a data set in the system that has the same name as a Special Resource, but there is not necessarily any connection between the two. (See 9.1, “Dataset triggering” on page 204.)

A Special Resource has two sorts of status: *availability*, so it can be available or not available, and it has *allocation*, which can be in use or not. A Special Resource can also have a quantity, but that is not used in relation to ETT.

Special Resource ETT processing occurs when a Special Resource that has been specified as an ETT Trigger is set to Available using means other than the Tivoli Workload Scheduler for z/OS dialog or PIF.

You can specify in the ETT definition (using the Actual Status Flag) whether triggering is to occur every time a request is received by the Controller to set the Special Resource to available status, regardless of whether it is already available or whether there must be an Actual Status change from not available to available for ETT processing to occur.

All Special Resources that are to be used as ETT Triggers should always be defined in the Special Resource database (option 1.6) with a quantity of one and a default availability of No. It is also recommended that they be used *only* for control. These settings will avoid unexpected processing, and strange but harmless occurrences in the Daily Plan Report. It is also recommended that you do not rely on DYNAMICADD to create these Special Resources.

Figure 9-15 on page 233 shows the Special Resource ETT demos, with the same applications as the Job Triggers in Figure 9-14 on page 231. There is one operation, on a WTO workstation.

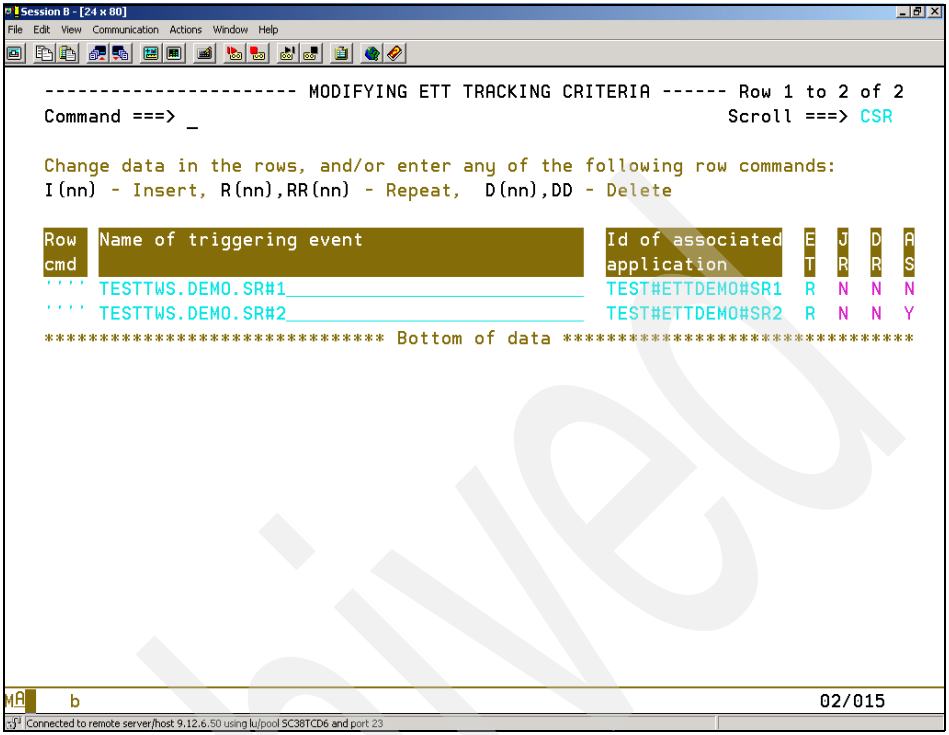


Figure 9-15 Special Resource ETT demos

1. When we submit a job to set TESTTWS.DEMO.SR#1 to Available, regardless of that resource's current status, an instance of TEST#ETTDEMO#SR1 will be added into the current plan.
2. When we submit a job to set TESTTWS.DEMO.SR#2 to Available, TEST#ETTDEMO#SR2 will be added only if TESTTWS.DEMO.SR#2 is *not* available when it starts.

SRSTAT processing can be very secure. Authority checking is done by the integrated Tivoli Workload Scheduler for z/OS interface with the security package (RACF, ACF/2, or TopSecret) via the SR.SRNAME Tivoli Workload Scheduler for z/OS subresource. Checking is done by the Tracker subsystem to which the SRSTAT is directed, so you must code an AUTHDEF init statement in the Tracker INIT PARMS, and the security profiles must be available on each system where you have a Tivoli Workload Scheduler for z/OS Tracker.

10

Tivoli Workload Scheduler for z/OS variables

This chapter provides information about how to set up Tivoli Workload Scheduler for z/OS variables, and includes many illustrations to guide you through the process along with some other detail examples not found in the Tivoli Workload Scheduler for z/OS guides. When you are finished with this chapter, you should be comfortable in the basics of Tivoli Workload Scheduler for z/OS variables and with user-defined variables.

We cover these topics in this chapter:

- Variable substitution
- Tivoli Workload Scheduler for z/OS supplied JCL variables
- Tivoli Workload Scheduler for z/OS variable table
- Tivoli Workload Scheduler for z/OS variables on the run

10.1 Variable substitution

Tivoli Workload Scheduler for z/OS supports automatic substitution of variables during job setup and at job submit. Tivoli Workload Scheduler for z/OS has many standard variables, and a listing of these variables can be found in the “Job Tailoring” chapter of the *IBM Tivoli Workload Scheduler for z/OS Managing the Workload Version 8.2,* SC32-1263.

You can also create your own variables by using the OPC JCL variable tables (option =1.9) shown in Figure 10-1. When you create your own variables, they are stored in the variable tables in the Tivoli Workload Scheduler for z/OS database. This ability is also unique, because it gives you the ability to have the same variable name in different variable tables and thus make the value different for each associated job you may be using it for.

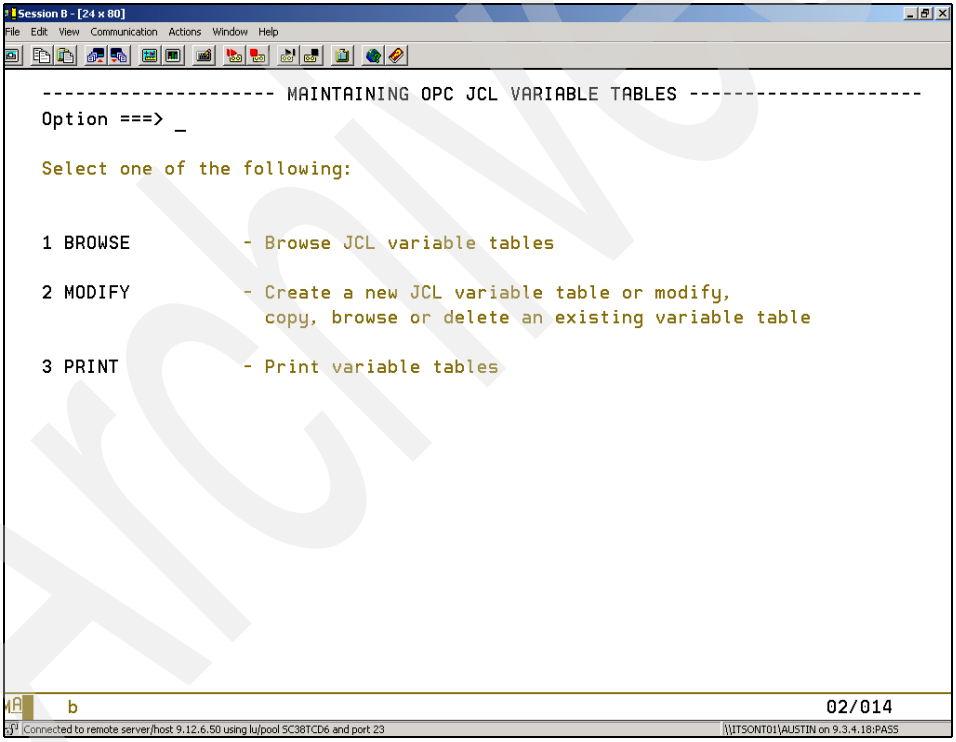


Figure 10-1 JCL Variable Table Option =1.9

In Tivoli Workload Scheduler, you can create variables in job statements, comment statements, and in any in-stream data within the job. The limitations to Tivoli Workload Scheduler for z/OS variables is that you cannot use them within cataloged or in-stream procedures. Any variable you have in a comment is

substituted, even variables to the right of the job statement. When you create a variable in a table, you are required to specify whether it should substitute at job setup at job submission or both. You can even have the variable setup as a promptable variable to allow interaction for the user to modify the variable prior to submission. It is important to remember that if you have the same variable name in different tables, make sure the right concatenation is in effect when the substitution occurs. In Example 10-1, variable TWSTEST is in the JCL twice, but the user has it defined in two separate tables, thus with two separate values. To do this we essentially do a call to the table where the variable is defined to resolve the variable properly. In the example, DDNAME1 did a call to TABLE2 for the TWSTEST variable, and DDNAME2 had a call for TABLE1 for the TWSTEST variable. This can be done throughout the JCL. It is important to make the call to the right variable table.

Example 10-1 Sample JCL for table search

```
//*%OPC SCAN
//*%OPC SEARCH NAME=(TABLE2)
//DDNAME1 DD DSN=&TWSTEST..FINANCE,DISP=SHR
//*%OPC TABLE NAME=(TABLE1)
//DDNAME2 DD DSN=&TWSTEST.&DATASET2.,DISP=SHR
```

10.1.1 Tivoli Workload Scheduler for z/OS variables syntax

Tivoli Workload Scheduler for z/OS variables have three different starting points, as shown in Example 10-2. One is with an ampersand (&), which instructs Tivoli Workload Scheduler for z/OS to resolve the variable from left to right. Second, a percent sign (%) does just the opposite of the ampersand; it resolves from right to left. Finally, a question mark (?) is used for tabulation of the variable.

Example 10-2 Sample variable syntax

```
&VARTWS1&VARTWS2
&VARTWS2%VARTWS2
?1OVARTWS3
```

Variables can also be terminated using the same variable syntax (&, ? or %). They can also be terminated by using a comma (,), parenthesis (), a blank (b), forward slash (/), single quote (‘), asterisk (*), double ampersand (&&), plus sign (+), dash or minus sign (-), an equals sign (=), or a period.

Example 10-3 Example of variable termination

```
&VARTWS &VARTWS2..REDBOOK
//DDNAME DD DSN=&VAR1..&VAR2(&VAR3)
```

Here is how the Tivoli Workload Scheduler for z/OS Controller parms should look. Under VARSUB in Example 10-4, there are three options:

SCAN (default) Enables you to have Tivoli Workload Scheduler for z/OS scan for variables only if a //*%OPC SCAN directive is defined in the JCL. (Note: Sometimes users like to put comments in their JCL, but using characters such as & or % in the comments could cause some problems with the job, so it is best to use SCAN in most cases.)

YES Permits Tivoli Workload Scheduler for z/OS to always scan for variables in all JCL that is submitted through Tivoli Workload Scheduler.

NO Tells Tivoli Workload Scheduler for z/OS to not scan for variables.

Example 10-4 Tivoli Workload Scheduler for z/OS Controller Parms

```
OPCOPTS  OPCHOST(YES)
         APPCTASK(NO)
         ERDRTASK(0)
         EWTRTASK(NO)
         GSTASK(5)
         JCCTASK(NO)
         NCFTASK(NO)
         RECOVERY(YES)       ARPARM(STDAR)
         RODMTASK(NO)
         VARSUB(SCAN)        GTABLE(GLOBAL)
```

To take from the sample above when we have VARSUB(SCAN) defined in the Controller Parms, we must use the //*%OPC SCAN JCL directive to start variable substitution. Example 10-5 shows a sample JCL of the OPC SCAN directive. There appears to be a variable called &MODULE right before the OPC SCAN directive. The &MODULE will not be substituted because it comes before the SCAN, but &LIBRARY will be resolved as it appears after the SCAN directive.

Example 10-5 Sample OPC SCAN directive

```
//TWSTEST JOB (REDBOOK),'Directive',CLASS=A
//STEP1 EXEC PGM=&MODULE.
//*%OPC SCAN
//STEPLIB DD DSN=TWS.LOAD.&LIBRARY.,DISP=SHR
//EQQMLIB DD DSN=TWS.MESSAGE.LIBRARY,DISP=SHR
//EQQMLOG DD SYSOUT=A
//SYSIN DD *
/*
```

10.2 Tivoli Workload Scheduler for z/OS supplied JCL variables

There are many Tivoli Workload Scheduler for z/OS supplied variables, which can be found in the “Job Tailoring” chapter of *IBM Tivoli Workload Scheduler Managing the Workload Version 8.2,* SC32-1263. Table 10-1 lists some of those variables.

Table 10-1 Occurrence-related supplied variables

Variable name	Length (in bytes)	Description
OADID	16	Application ID
OADOWNER	16	Occurrence owner
OAUGROUP	8	Authority group
OCALID	16	Calendar name
ODAY	1	Occurrence input arrival day of the week; 1-7 with 1 for Monday, 7 for Sunday...
ODD	2	Occurrence input arrival day of month, in DD format
ODDD	3	Occurrence input arrival day of the year, in DDD format
ODMY1	6	Occurrence input arrival date in DDMMYY format
ODMY2	8	Occurrence input arrival date in DD/MM/YY format
OFREEDAY	1	Denotes whether the occurrence input arrival date is a freeday (F) or workday (W)
OHH	2	Occurrence input arrival hour in HH format
OHHMM	4	Occurrence input arrival hour and minute in HHMM format
OMM	2	Occurrence input arrival month in MM format
OMMYY	4	Occurrence input arrival month and year in MMYY format
OWW	2	Occurrence input arrival week of the year in WW format
OWWD	3	Occurrence input arrival week, and day within week, in WWD format, where WW is the week number within the year, and D is the day within the week

Variable name	Length (in bytes)	Description
OWWLAST	1	A value, Y (yes) or N (no), that indicates whether the occurrence input arrival date is in the last week of the month
OWWMONTH	1	A value between 1 and 6 that indicates the occurrence input arrival week-in-month, where each new week begins on a Monday. For example, consider these occurrence input arrival dates for the month of March in 1997: DATE/OWWMONTH Saturday 1st/1 Monday 3rd/2 Monday 31/6
OYMD	8	Occurrence input arrival date in YYYYMMDD format
OYM	6	Occurrence input arrival month within year in YYYYMM format
OYMD1	6	Occurrence input arrival date in YYMMDD format
OYMD2	8	Occurrence input arrival date in YY/MM/DD format
OYMD3	10	Occurrence input arrival date in YYYY/MM/DD format
OYY	2	Occurrence input arrival year in YY format
OYYDDD	5	Occurrence input arrival date as a Julian date in YYDDD format
OYYMM	4	Occurrence input arrival month within year in YYMM format
OYYYY	4	Occurrence input arrival year in YYYY format

The book also lists date-related supplied variables, operation-related supplied variables, and dynamic-format supplied variables.

10.2.1 Tivoli Workload Scheduler for z/OS JCL variable examples

For a simple variable that displays the current system date in MMDDYY format, Example 10-6 on page 241 shows the initial setup prior to job submission, and Example 10-7 on page 241 is the resolution of the variable.

Example 10-6 Current system date

```
//TWSTEST1 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*%OPC SCAN
//*********************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*********************************************************
//STEP10 EXEC IEFBR14
//*
//        CURRDATE='&CMM&CDD&CYY'            CURRENT SYSTEM DATE
//*
```

In Example 10-7 the OPC SCAN has changed the % to a >, indicating that the scan was resolved successfully. If there were a problem with the variable, a OJCV error would occur and the job would be put in the error list.

Example 10-7 Current system date resolved

```
//TWSTEST1 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*>OPC SCAN
//*********************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*********************************************************
//STEP10 EXEC IEFBR14
//*
//        CURRDATE='091405'            CURRENT SYSTEM DATE
//*
```

In Example 10-8, we keep the current variable and add another that will show us both the current system date and the input arrival date of the job.

Example 10-8 Input arrival date and current system date

```
//TWSTEST2 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*%OPC SCAN
//*********************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*********************************************************
//STEP10 EXEC IEFBR14
//*
//*
//        IATDATE='&CMM&CDD&CYY'             TWS INPUT ARRIVAL DATE
//*
```

```
//        CURRDATE='&CMM&CDD&CYY'            CURRENT SYSTEM DATE
//*
```

Example 10-9 shows the variables resolved. The input arrival date is different from the current system date, which means that this job's application was brought in or scheduled into the current plan on the prior day.

Example 10-9 Input arrival date and current system date resolved

```
//TWSTEST2 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*>OPC SCAN
//*********************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*********************************************************
//STEP10 EXEC IEFBR14
//*
//*
//        IATDATE='091305'            TWS INPUT ARRIVAL DATE
//*
//        CURRDATE='091405'           CURRENT SYSTEM DATE
//*
```

Example 10-10 uses a lot more variables at one time. Here we show the occurrence date, the occurrence input arrival time, current date, current input arrival time, application ID, owner, operation number, day of the week represented by a numeric value (1 for Monday...7 for Sunday), the week number in the year, the freeday, and the calendar the application or job is defined to.

Example 10-10 Multiple variable samples

```
//TWSTEST3 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*********************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*********************************************************
//STEP10 EXEC IEFBR14
//STEP20 EXEC IEFBR14
//*
//*%OPC SCAN
//*
//*********************************************************
//*  TWS DATE = &OMM/&ODD/&OYY  IATIME = &OHHMM
//*  CUR DATE = &CMM/&CDD/&CYY  IATIME = &CHHMM
//*
```

```
//* ADID = &OADID  OWNER = &OADOWNER  OPNO = &OOPNO
//* DAY  =&ODAY WEEK = &OWW FREEDAY = &OFREEDAY CAL = &OCALID
//*************************************************************
//*
```

Example 10-11 Multiple variable samples

```
//TWSTEST3 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*************************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*************************************************************
//STEP10 EXEC IEFBR14
//STEP20 EXEC IEFBR14
//*
//*>OPC SCAN
//*
//*************************************************************
//*  TWS DATE = 09/13/05  IATIME = 0700
//*  CUR DATE = 09/14/05  IATIME = 1025
//*
//* ADID = TWSREDBOOK  OWNER = OPER  OPNO = 010
//* DAY  =3 WEEK = 37 FREEDAY = W CAL = DEFAULT
//*************************************************************
//*
```

Example 10-12 shows how a temporary variable works in Tivoli Workload Scheduler. Temporary variables must start with a T to avoid an error. The format below has the Dynamic Format in MM/DD/YY. *IBM Tivoli Workload Scheduler Managing the Workload Version 8.2,* SC32-1263 lists other formats for OCDATE. Example 10-12 shows still more, such as &ODD, &ODAY, &OWW, and &OYY. Other valid expression types are WD, WK, MO, and YR.

Example 10-12 Temporary variable

```
//TWSTEST4 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*%OPC SCAN
//*%OPC SETFORM OCDATE=(MM/DD/YY)
//*%OPC SETVAR  TVAR=(OCDATE-5CD)
//*************************************************************
//* TESTING TWS JCL TEMPORARY VARIABLE SUBSTITUTION
//* REMEMBER ALL TEMPORARY VARIABLES MUST START WITH A 'T'
//*************************************************************
//STEP10 EXEC IEFBR14
```

```
//*
OLDDATE='&TVAR'
//*
//*******************************************************
```

In Example 10-13, the occurrence date for this job was 09/14/05. The SETVAR set the value for TVAR to take the occurrence day and subtract five calendar days. Thus, the result is 09/09/05.

Example 10-13 Temporary variable resolved

```
//TWSTEST4 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*>OPC SCAN
//*>OPC SETFORM OCDATE=(MM/DD/YY)
//*>OPC SETVAR  TVAR=(OCDATE-5CD)
//*******************************************************
//* TESTING TWS JCL TEMPORARY VARIABLE SUBSTITUTION
//* REMEMBER ALL TEMPORARY VARIABLES MUST START WITH A 'T'
//*******************************************************
//STEP10 EXEC IEFBR14
//*
OLDDATE='09/09/05'
//*
```

Example 10-14 uses multiple temporary variables and multiple SEFORMS using the same Dynamic Format variable. Note that we use the OCDATE twice and each preceding occurrence of the SETFORM overrides the prior SETFORM, so doing the SETFORM OCDATE=(CCYY) overrides the OCDATE=(MM) for any further SETVAR we use.

Example 10-14 Multiple temporary variables

```
//TWSTEST5 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*%OPC SCAN
//*%OPC SETFORM OCDATE=(MM)
//*%OPC SETVAR  TMON=(OCDATE-1MO)
//*%OPC SETFORM OCDATE=(CCYY)
//*%OPC SETVAR  TYEAR=(OCDATE-1MO)
//*******************************************************
//* TESTING TWS JCL TEMPORARY VARIABLE SUBSTITUTION
//* REMEMBER ALL TEMPORARY VARIABLES MUST START WITH A 'T'
//*******************************************************
//STEP10 EXEC IEFBR14
```

```
//   MONTH=&OMM,       CURRENT MONTH
//   YEAR=&OYYYY       CURRENT YEAR
//STEP20 EXEC IEFBR14
//   MONTH=&TPMON,     PRIOR MONTH
//   YEAR=&TYEAR       PRIOR YEAR
//*
```

In the resolution of the multiple temporary variables in Example 10-15, you see the SETFORMs and SETVARs; also, we use the same type of subtraction but for TPYY, we subtract by 10 months. The result is the prior month and prior year and current month and current year.

Example 10-15 Multiple temporary variables resolved

```
//TWSTEST5 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*>OPC SCAN
//*>OPC SETFORM OCDATE=(MM)
//*>OPC SETVAR  TMON=(OCDATE-1MO)
//*>OPC SETFORM OCDATE=(CCYY)
//*>OPC SETVAR  TYEAR=(OCDATE-10MO)
//*************************************************************
//* TESTING TWS JCL TEMPORARY VARIABLE SUBSTITUTION
//* REMEMBER ALL TEMPORARY VARIABLES MUST START WITH A 'T'
//*************************************************************
//STEP10 EXEC IEFBR14
//   MONTH=09,       CURRENT MONTH
//   YEAR=2005       CURRENT YEAR
//STEP20 EXEC IEFBR14
//   MONTH=08,     PRIOR MONTH
//   YEAR=2004       PRIOR YEAR
//*
```

Example 10-16 gets into Tivoli Workload Scheduler for z/OS JCL Directives. Here we use the BEGIN and END actions to include selected in-line JCL statements or to exclude selected in-line JCL statements. We are using the COMP to compare what the current occurrence day is (&ODAY) and whether it is equal to or not equal to day 3, which is Wednesday.

Example 10-16 Tivoli Workload Scheduler for z/OS JCL directives

```
//TWSTEST6 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*%OPC SCAN
//*************************************************************
```

```
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*******************************************************
//*%OPC BEGIN ACTION=INCLUDE,PHASE=SUBMIT,
//*%OPC       COMP=(&ODAY..NE.3)
//*
//STEP10 EXEC IEFBR14
//*%OPC END ACTION=INCLUDE
//*
//*%OPC BEGIN ACTION=INCLUDE,PHASE=SUBMIT,
//*%OPC       COMP=(&ODAY..EQ.3)
//*
//STEP20 EXEC IEFBR14
//*
//*%OPC END ACTION=INCLUDE
//*
```

As you can see in Example 10-17, when the variable resolves, it is in a sense looking for a match. In the first begin action, Wednesday, which is represented by 3, is not equal (NE) to itself, so this comparison is false. The next begin action shows the comp (compare) of 3 is equal to 3, thus stating that if today is Wednesday perform Step20, which is true.

Example 10-17 Tivoli Workload Scheduler for z/OS JCL directives resolved

```
//TWSTEST6 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*>OPC SCAN
//*******************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*******************************************************
//*>OPC BEGIN ACTION=INCLUDE,PHASE=SUBMIT,
//*>OPC       COMP=(3.NE.3)
//*
//*>OPC BEGIN ACTION=INCLUDE,PHASE=SUBMIT,
//*>OPC       COMP=(3.EQ.3)
//*
//STEP20 EXEC IEFBR14
//*
//*>OPC END ACTION=INCLUDE
//*
```

Example 10-18 shows a begin action and comp against a specified date. So if the date is equal to 050914, then TWSTEST will be included.

Example 10-18 Date compare

```
//TWSTEST7 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*%OPC SCAN
//*********************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*********************************************************
//STEP010 EXEC IEFBR14
//STEP020 EXEC IEFBR14
//*%OPC BEGIN ACTION=INCLUDE,PHASE=SUBMIT,
//*%OPC       COMP=(&OYMD1..EQ.050914)
//TWSTEST DD DISP=SHR,DSN=TWS.TEST.VAR
//*%OPC END ACTION=INCLUDE
```

Example 10-19 shows the resolved date compare, and the dates match so TWSTES is included. If the dates did not match, TWSTEST would be excluded from the JCL because the COMP would be false.

Example 10-19 Date compare resolved

```
//TWSTEST7 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*>OPC SCAN
//*********************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION
//*********************************************************
//STEP010 EXEC IEFBR14
//STEP020 EXEC IEFBR14
//*>OPC BEGIN ACTION=INCLUDE,PHASE=SUBMIT,
//*>OPC       COMP=(050914.EQ.050914)
//TWSTEST DD DISP=SHR,DSN=TWS.TEST.VAR
//*>OPC END ACTION=INCLUDE
```

Example 10-20 has multiple compares against particular dates. If any of those comparisons are true, then TESTTWS=## will be included.

Example 10-20 Multiple comparisons

```
//TWSTEST8 JOB (290400),'TEST VARS',CLASS=A,MSGCLASS=H
//*%OPC SCAN
//STEP010 EXEC IEFBR14
//*%OPC BEGIN ACTION=INCLUDE,COMP=(&OYMD1..EQ.050911)
//   TESTTWS=01
//*%OPC END ACTION=INCLUDE
//*%OPC BEGIN ACTION=INCLUDE,COMP=(&OYMD1..EQ.050912)
```

```
//   TESTTWS=02
//*%OPC END ACTION=INCLUDE
//*%OPC BEGIN ACTION=INCLUDE,COMP=(&OYMD1..EQ.050913)
//   TESTTWS=03
//*%OPC END ACTION=INCLUDE
//*%OPC BEGIN ACTION=INCLUDE,COMP=(&OYMD1..EQ.050914)
//   TESTTWS=04
//*%OPC END ACTION=INCLUDE
//*%OPC BEGIN ACTION=INCLUDE,COMP=(&OYMD1..EQ.050915)
//   TESTTWS=05
//*%OPC END ACTION=INCLUDE
//*
```

Example 10-21 shows the resolved comparisons from Example 10-20 on page 247. For each false comparison the JCL is omitted, as it is not needed to submit the JCL. The true comparison thus shows TESTTWS=04 will be included.

Example 10-21 Multiple comparisons resolved

```
//TWSTEST8 JOB (290400),'TEST VARS',CLASS=A,MSGCLASS=H
//*>OPC SCAN
//STEP010 EXEC IEFBR14
//*>OPC BEGIN ACTION=INCLUDE,COMP=(050914.EQ.050911)
//*>OPC BEGIN ACTION=INCLUDE,COMP=(050914.EQ.050912)
//*>OPC BEGIN ACTION=INCLUDE,COMP=(050914.EQ.050913)
//*>OPC BEGIN ACTION=INCLUDE,COMP=(050914.EQ.050914)
//   TESTTWS=04
//*>OPC END ACTION=INCLUDE
//*>OPC BEGIN ACTION=INCLUDE,COMP=(050914.EQ.050915)
//*
```

As previously discussed, there may be comments in the JCL that use a & or %. This can cause some problems in the JCL when an OPC SCAN is set up. TWS will likely treat these as potential variables, thus not recognize them and cause an OJCV error. To correct this you can either omit these characters from the JCL comments or wrap OPC BEGIN ACTION=NOSCAN around the comments, then you can terminate the NOSCAN by issuing a END ACTION=NOSCAN as you would in your typical BEGIN ACTION=INCLUDE, but here you are telling Tivoli Workload Scheduler for z/OS not to scan this part of the JCL for TWS variables.

10.3 Tivoli Workload Scheduler for z/OS variable table

You can define a global variable table. The name of this variable table is specified in the GTABLE keyword of the initialization statement OPCOPTS (refer to *IBM Tivoli Workload Scheduler Customization and Tuning Version 8.2,* SC32-1265). If Tivoli Workload Scheduler for z/OS cannot find a variable in the variable tables specified for the operation or in the operation job, it searches the global variable table. The order for which a table is searched for a variable is based on the application or operation setup: the SEARCH TABLE directive in the JCL, followed by the application table (if it exists), then the global table.

For example, Figure 10-2 shows the TWSVAR assigned for this application. When the job starts, it searches this table for specific variables assigned in the JCL. If it does not find a variable in there, it will search for it if there is a //*%OPC SEARCH NAME=(TABLE1). It searches TABLE1, and if the variable is not defined there, it goes to the application table, followed by the global table.

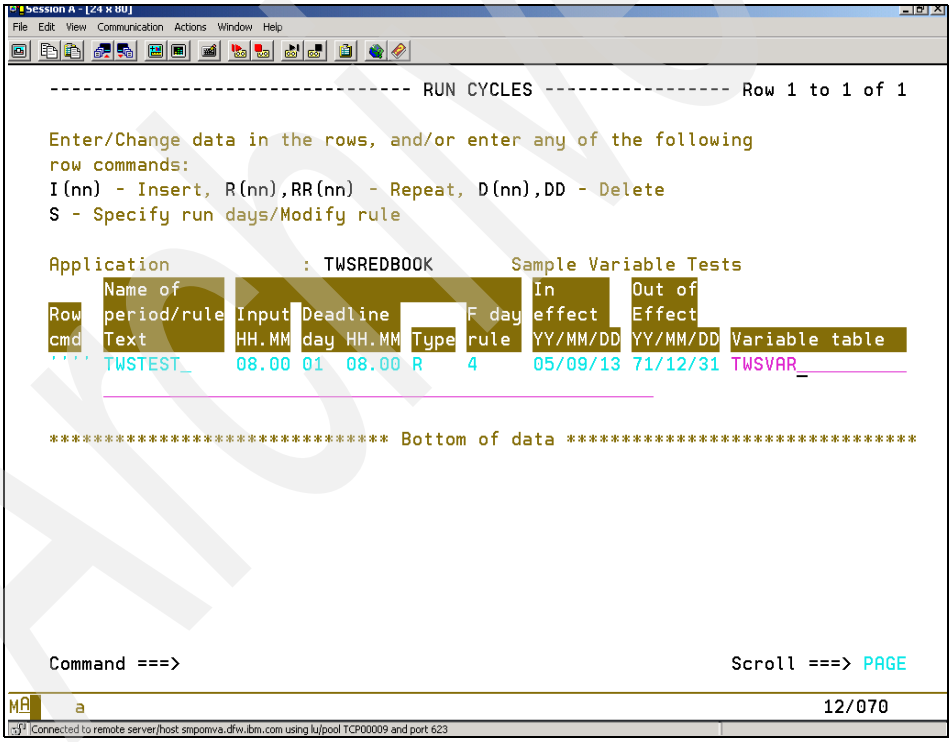


Figure 10-2 Defining a variable table for a Run Cycle

10.3.1 Setting up a table

To set up a variable table, use option =1.9 from almost anywhere in Tivoli Workload Scheduler. As shown in Figure 3-3, you can browse, modify, or print. Option 2 offers the ability to choose a variable table name, a variable name, or owner.

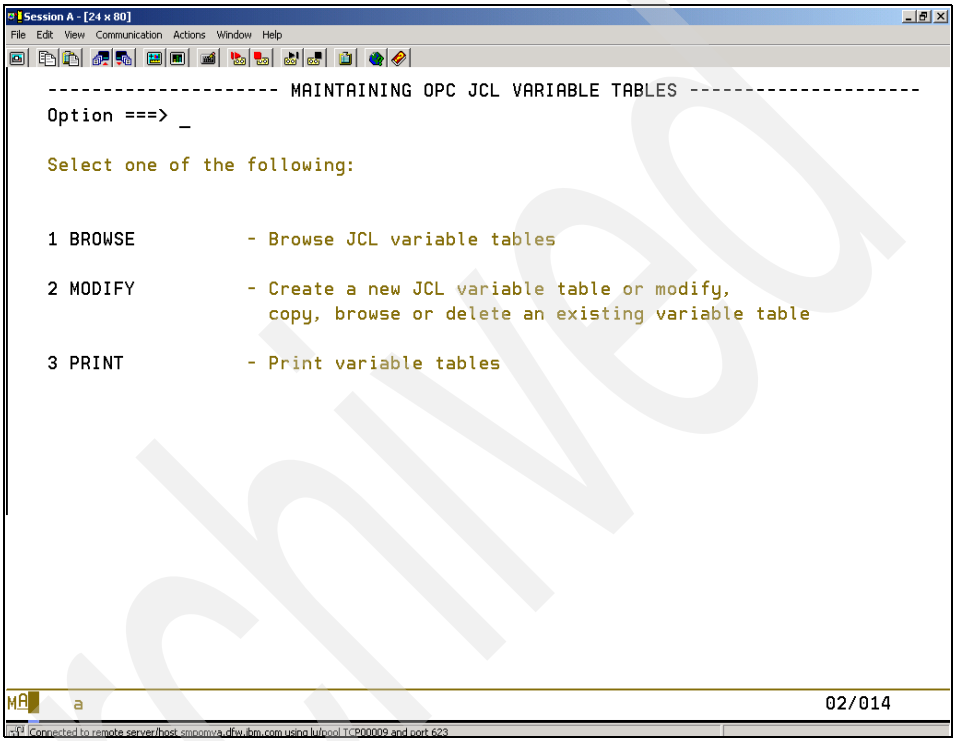


Figure 10-3 Maintaining OPC JCL Variable Tables

You can also use wildcards to narrow your search or leave each field blank to see all the tables defined.

Figure 10-4 shows the list of JCL variable tables. From here we can browse, modify, copy, or delete the existing tables or create a new table.

Note: Here it would not be wise to delete a table, especially the global table, so Tivoli Workload Scheduler for z/OS will prompt you to confirm that action.

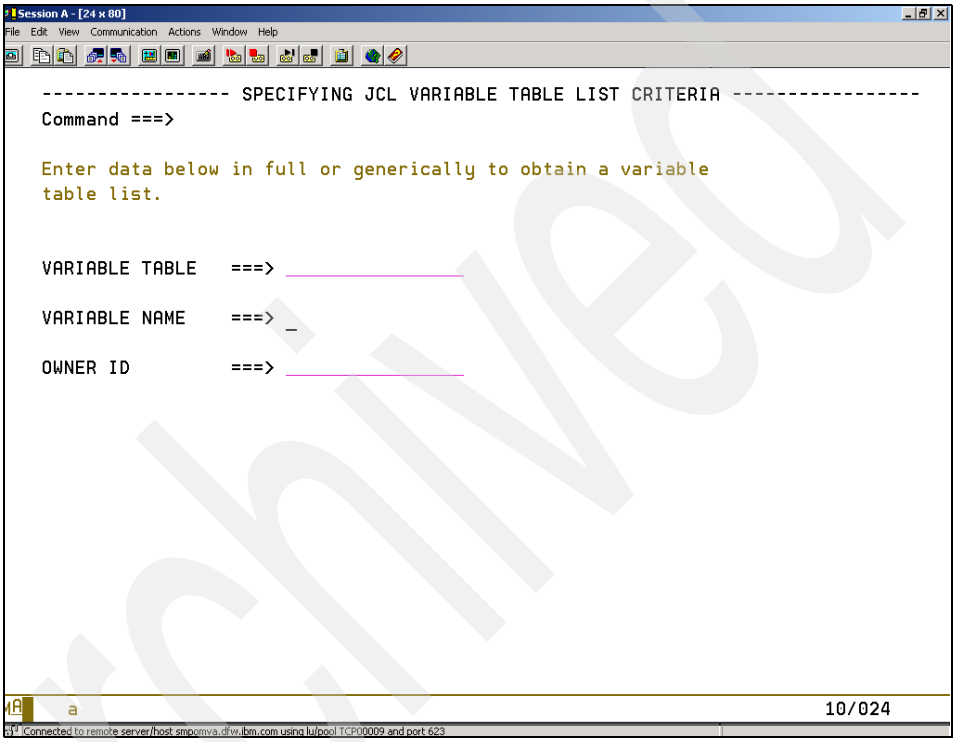


Figure 10-4 Specifying JCL Variable Table List Criteria

Figure 10-5 shows how to create a table by using the Create command.

- Choose a unique or generic table name, 1 - 16 alphanumeric characters long.
- Owner ID can be from 1 to 16 characters long.
- The Table Description field is optional and can be up to 24 characters in length. The variable name can be from 1-8 characters, but the first character must be alphabetic or national.
- Subst. Exit can be 1 - 8 alphanumeric characters in length with the first character alphabetic or national. This is optional and used for an exit name that can validate or set the variable, or both.
- In the Setup field, also optional, determine how the variable substitution should be handled. If you set N, which is the default, the variable will be substituted at submission. Y is similar to N in the sense that there will be no interaction, but the variable will be substituted at submission if setup is not performed for the operation. P is for interaction to take place at job setup (see 10.3.2, “Creating a promptable variable” on page 256 for setting up a promptable variable).

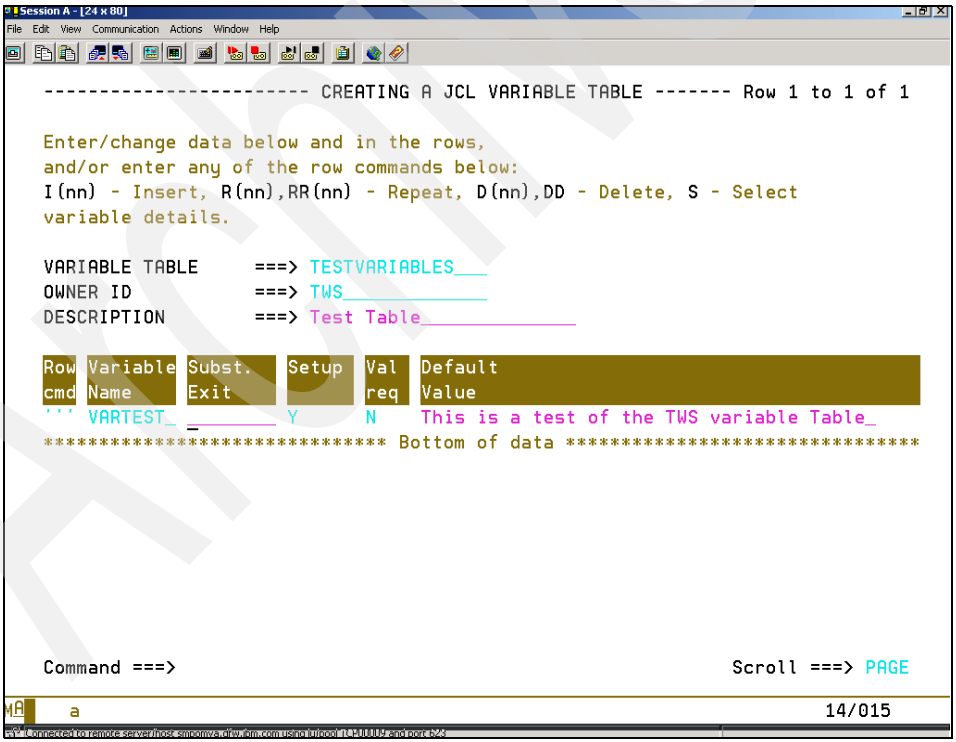


Figure 10-5 Creating a JCL Variable Table

Now we put our variable table to the test. Example 10-22 shows how to set up the JCL with the Search Table Name in the JCL. Recall that we can also add it in the run cycle, and you can add the variable table in option =5.1, “Adding applications to the Current Plan Panel”. We indicate the Search Name for the table as shown, but we could also add up to 16 tables in this way and include the global and application tables as well. Otherwise, Tivoli Workload Scheduler for z/OS will search TWSTESTVAR table for the VARTEST variable. If it is not found, it would search in the application table, then the global table. If the variable was not found in any of the tables, the job would end with an OJVC error.

Example 10-22 Variable substitution from a table

```
//TWSTEST9 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*%OPC SCAN
//*%OPC SEARCH NAME=(TWSTESTVAR)
//*************************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION FROM A TABLE
//*************************************************************
//STEP10 EXEC IEFBR14
//*
//%VARTEST
//*
```

Example 10-23 shows the resolution of the variable that we have defined.

Example 10-23 Variable substitution from a table resolved

```
//TWSTEST9 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
//         MSGCLASS=H,TYPRUN=SCAN
//*>OPC SCAN
//*>OPC SEARCH NAME=(TWSTESTVAR)
//*************************************************************
//* TESTING TWS JCL VARIABLE SUBSTITUTION FROM A TABLE
//*************************************************************
//STEP10 EXEC IEFBR14
//*
//This is a Variable Table Test
//*
```

This results in an error. Figure 10-6 shows a simple example of what an OJCV error looks like from the =5.4 panel. Enter a J on the command line and press Enter to look at the JCL for the problem. Tivoli Workload Scheduler for z/OS does a good job in pointing out where the problem is located. When Tivoli Workload Scheduler for z/OS scans the JCL for variables, it scans from top to bottom.

If there are several variables in the JCL, there may or may not be other problems even after you fix the initial error. If you understand and fix the problem, restart the job and get another OJCV to see whether there is another variable problem. Always remember that when you update the job in the current plan, you have to update the production JCL to ensure that this problem does not reoccur in the next scheduled run of this job.

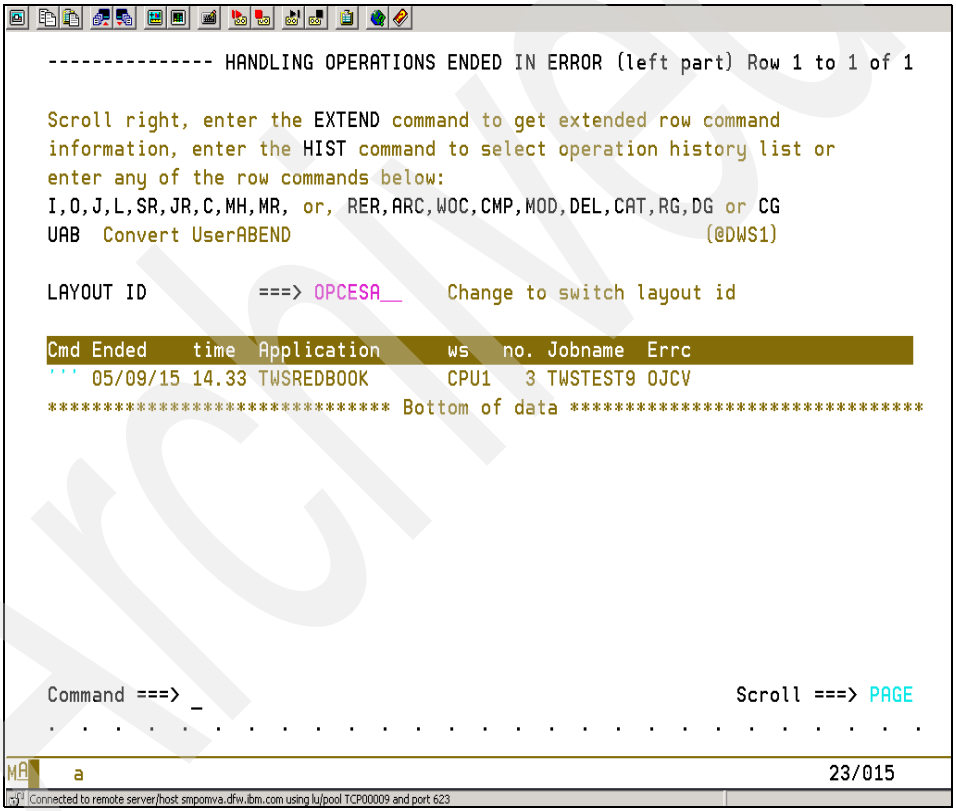


Figure 10-6 OJCV Error in 5.4 Panel

Example 10-24 shows what happens when an OJCV occurs. When you give the J row command from 5.4 for the operation or job, you will see a NOTE that is highlighted to indicate a Variable Substitution Failed. In this case, it says that the problem occurred on line 10 of the JCL.Based on Example 10-22 on page 253, line 10 shows that the variable name is misspelled, so Tivoli Workload Scheduler for z/OS could not find the variable in the tables and forced the job in error.

Example 10-24 Output of the job

```
000001 //TWSTEST9 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
000002 //         MSGCLASS=H,TYPRUN=SCAN
000003 //*%OPC SCAN
000004 //*%OPC SEARCH NAME=(TWSTESTVAR)
000005 //*************************************************************
000006 //* TESTING TWS JCL VARIABLE SUBSTITUTION FROM A TABLE
000007 //*************************************************************
000008 //STEP10 EXEC IEFBR14
000009 //*
000010 //%vartets
000011 //*
=NOTE= VARIABLE SUBSTITUTION FAILED.
====== //TWSTEST9 JOB (290400),'TEST VARS',CLASS=A,MSGLEVEL=(1,1),
====== //         MSGCLASS=H,TYPRUN=SCAN
====== //*>OPC SCAN
====== //*>OPC SEARCH NAME=(TWSTESTVAR)
====== //*************************************************************
====== //* TESTING TWS JCL VARIABLE SUBSTITUTION FROM A TABLE
====== //*************************************************************
====== //STEP10 EXEC IEFBR14
====== //*
=NOTE= //*>EQQJ535E 09/15 14.39.31
=NOTE= //*>        UNDEFINED VARIABLE vartets LINE 00010 OF ORIG JCL
====== //%VARTETS
====== //*
```

To resolve it, we simply correct the variable misspelling, use END to end the edit of the JCL, and then run a JR on the row command for a job restart. Then, the variables will be resolved and the job will be submitted to JES.

10.3.2 Creating a promptable variable

Along the same premise as creating a variable in a variable table, we can create a promptable variable:

1. From the setup option of the variable, we set P for Prompt, then set a Default Value. For example, we can have V12345 and the purpose of this promptable variable will be for a user to enter a volser number in this job.
2. You also have to create a setup workstation (unless it has already been done). Go to =1.1.2 of the Specifying Work Station List Criteria and press Enter. Then, run the CREATE command and enter the data shown in Figure 10-7. The workstation name here is JCL1 to indicate it is JCL related, but it can named to whatever you choose up to four alphanumeric characters with the first being alphabetic or national.

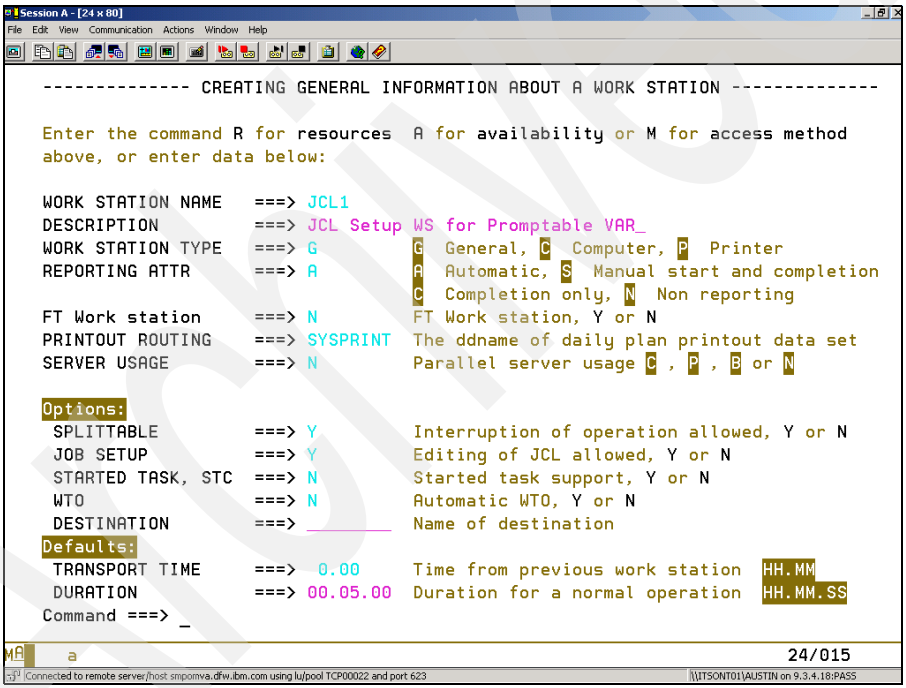


Figure 10-7 JCL setup workstation for promptable variable

The workstation type is General, the reporting attribute is Automatic, no FT Work Station, Printout Routing is the ddname where reports for this workstation should be printed, Server usage is Neither. Splittable is Yes so the operation can be interrupted, Job Setup is Yes because we will edit the JCL at startup, Started Task is set to No, WTO (Write To Operator) is set to No, and we do not need a Destination. No Transport Time is required, and Duration is set to 5

minutes but the time can be adjusted. For more about Duration, see *IBM Tivoli Workload Scheduler for z/OS Managing the Workload Version 8.2,* SC32-1263.

The command lines show three options: R for resources, A for availability, and M for access method, which is used for Tracker Agents.

3. Figure 10-8 shows a sample application of how the operations should be set up for a promptable variable in the Application Database. Note that we have the same jobname on two separate workstations. The first workstation, which we just set up, is called JCL1 because the operation of preparing a job must immediately be followed by the operation that runs the job on the computer workstation (CPU1). Also note that submit is set to N for this operation (JCL1), but set to Y for the CPU1 operation. As long as the operation is not waiting for other conditions (predecessors, special resources) to be met, the job can be started, as job setup is complete.

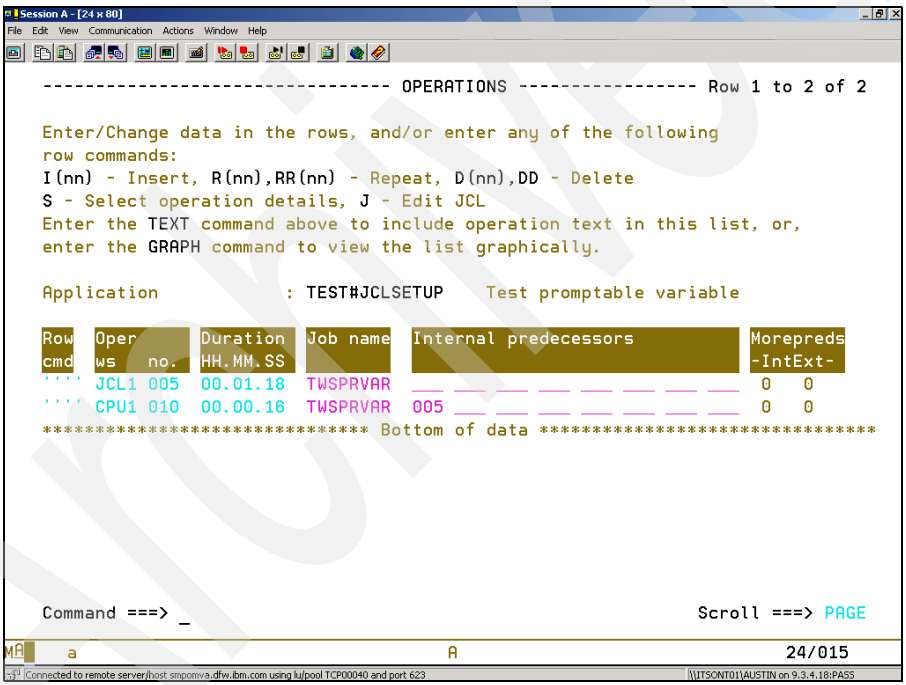


Figure 10-8 Operation setup

4. After the operations are set up and the application is in the current plan, we can work on the promptable variable in the Ready List (option =4.1). From here we can easily change the Workstation name to JCL1. Because we built this workstation for use with promptable variables, it will give us a list of all jobs in the Ready List that have a JCL1 workstation (Figure 3-9).

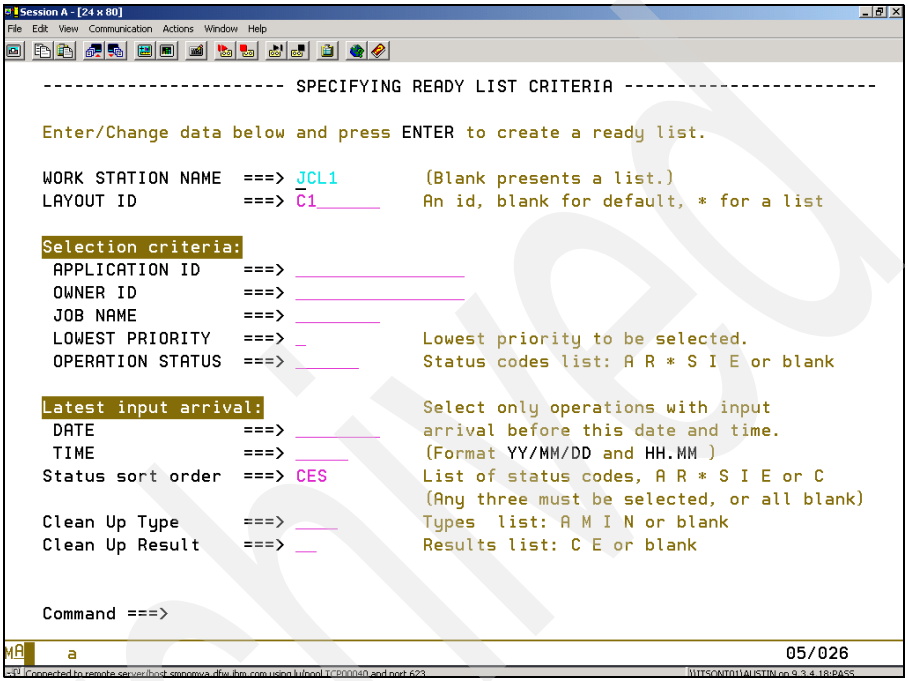


Figure 10-9 Specifying Ready List Criteria -entered workstation JCL1

5. The Ready List shows all operations that use the JCL1 workstation (Figure 10-10). We type N on the row command next to our operation to set the next logical status. When this occurs, Tivoli Workload Scheduler for z/OS will initiate this operation (setup operation) as S (started).

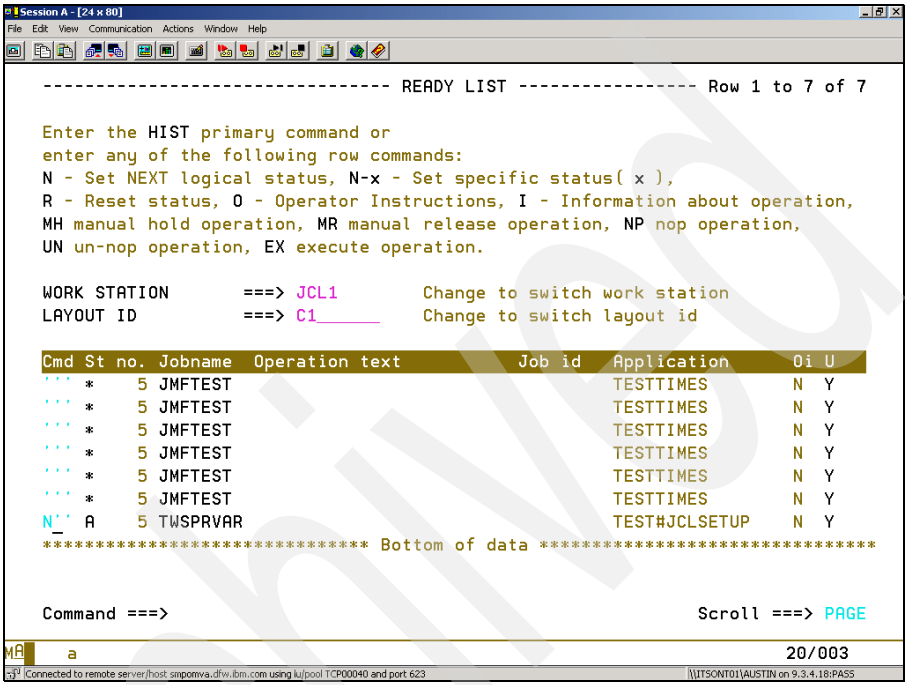


Figure 10-10 Setting Next Logical Status from the Ready List

6. The next action depends on whether we had a promptable variable in the job that has not been resolved. Because we do have such a variable, it is immediately put in edit mode to resolve the promptable variable (Figure 10-11). Here you see our variable name as it is defined in the JCL, and the default value that we assigned to it was v11111. All that has to be done is to edit that value as it should be.

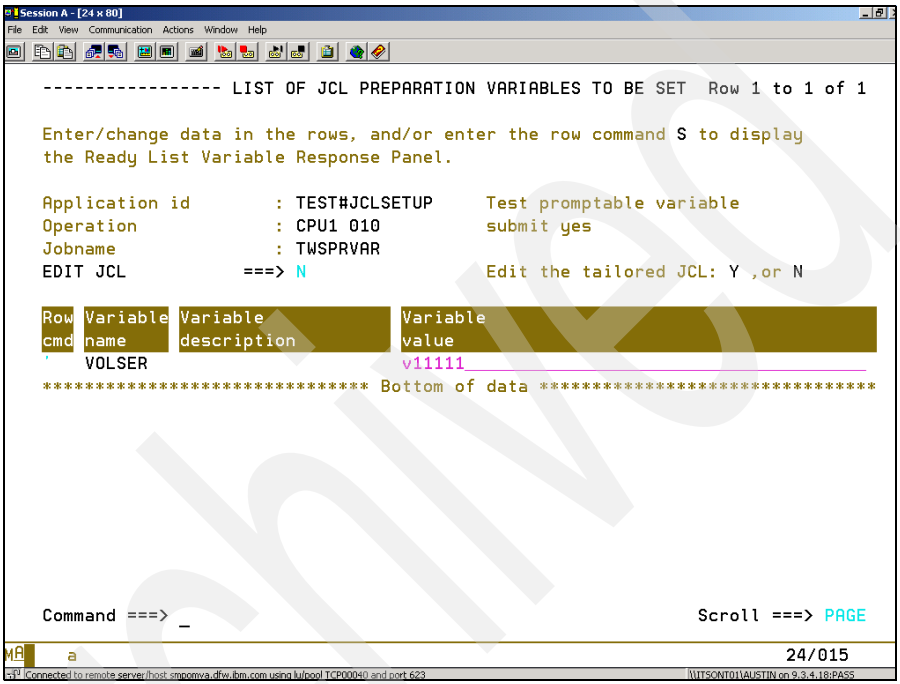


Figure 10-11 Ready List promptable variable

7. Therefore, we change the value as shown in Figure 3-12. Here you can see the change made to the variable. You could also type s on the row command and change the variable in the Variable Value field. Either way will work fine. Back out of the panel by pressing F3.

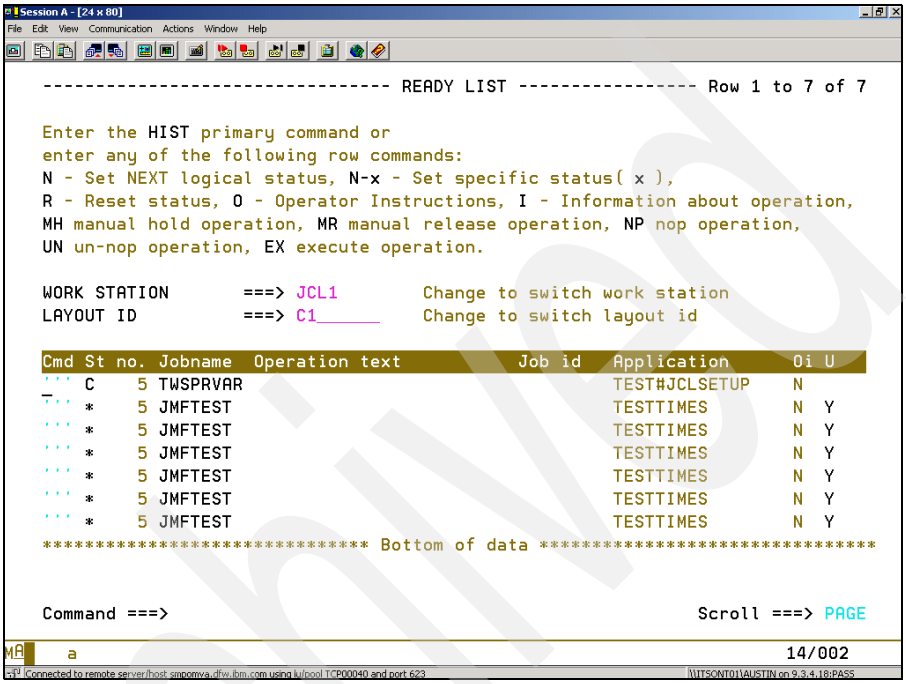


Figure 10-12 Edited variable

8. Press PF3, and our JCL1 operation is Complete (Figure 3-13). The job itself will start to run in Tivoli Workload Scheduler.

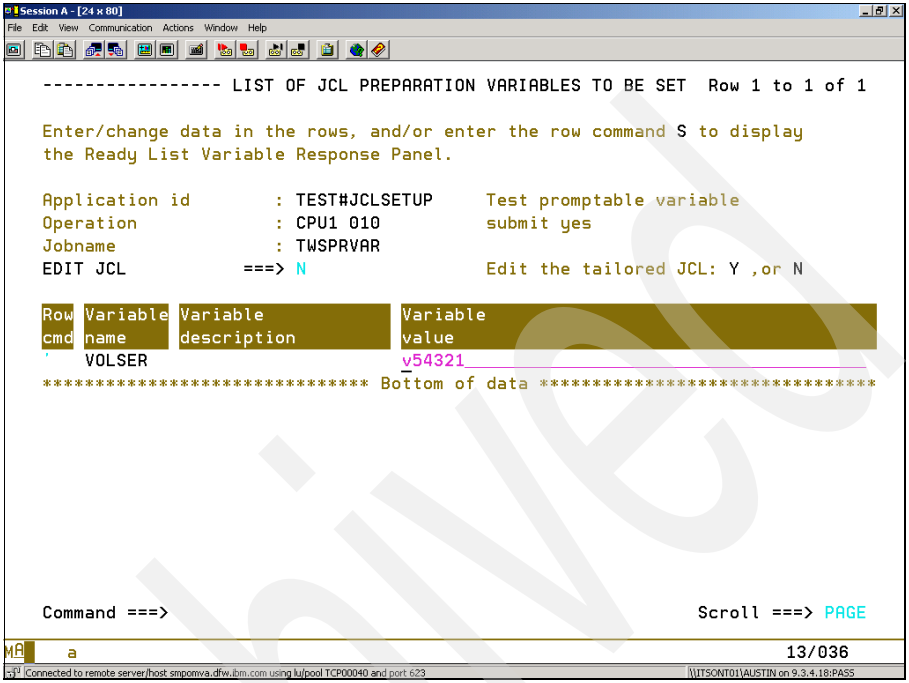


Figure 10-13 Complete JCL1 operation

Example 10-25 shows a “before” picture of the JCL, prior to when it started and the promptable variable was changed.

Example 10-25 Promptable variable job before

```
//TWSPRVAR JOB (),'PROMPTABLE JCL VAR ',
//*%OPC SCAN
//            CLASS=A,MSGCLASS=&HMCLAS
//*********************************************************************
//*  TWS TEST FOR A PROMPTABLE VARIABLE VOLSER=&VOLSER
//*********************************************************************
//*
//STEP10 EXEC IEFBR14
//*
```

Example 10-26 shows the “after” snapshot with the job completing successfully. The variable has changed to what we did in the previous sequence.

Example 10-26 Promptable variable job resolved

```
//TWSPRVAR JOB (),'PROMPTABLE JCL VAR ',
//*>OPC SCAN
//             CLASS=A,MSGCLASS=2
//*********************************************************************
//*  TWS TEST FOR A PROMPTABLE VARIABLE V0LSER=v54321
//*********************************************************************
//*
//STEP10 EXEC IEFBR14
//*
```

10.3.3 Tivoli Workload Scheduler for z/OS maintenance jobs

Tivoli Workload Scheduler for z/OS temporary variables can also be used in Tivoli Workload Scheduler for z/OS maintenance jobs to update dates in the Control Cards as necessary. The Current Plan Extend and Long-Term Plan jobs are examples of where variables can be used. Example 10-27 is the copy of the Long-term Plan Extend JCL and a temporary variable that can be used to extend it.

Example 10-27 Long-term Plan Extend JCL

```
//TWSLTPEX JOB (0),'B SMITH',MSGLEVEL=(1,1),REGION=64M,
//         CLASS=A,COND=(4,LT),MSGCLASS=X,TIME=1440,NOTIFY=&SYSUID
/*JOBPARM S=SC64
//DELETE   EXEC PGM=IEFBR14
//OLDLIST  DD DSN=TWSRES4.TWSC.LTEXT.LIST,
//         UNIT=3390,SPACE=(TRK,0),DISP=(MOD,DELETE)
//ALLOC    EXEC PGM=IEFBR14
//NEWLIST  DD DSN=TWSRES4.TWSC.LTEXT.LIST,
//         UNIT=3390,DISP=(,CATLG),SPACE=(CYL,(9,9),RLSE),
//         DCB=(RECFM=FBA,LRECL=121,BLKSIZE=12100)
//*********************************************************************
//*
//*   Licensed Materials - Property of IBM
//*   5697-WSZ
//*   (C) Copyright IBM Corp. 1990, 2003 All Rights Reserved.
//*   US Government Users Restricted Rights - Use, duplication
//*   or disclosure restricted by GSA ADP Schedule Contract
//*   with IBM Corp.
//*
//* LONG TERM PLANNING - EXTEND THE LONG TERM PLAN
//*********************************************************************
//LTEXTEND EXEC PGM=EQQBATCH,PARM='EQQLTMOA',REGION=4096K
//EQQMLIB  DD DISP=SHR,DSN=EQQ.SEQQMSG0
```

```
//EQQPARM  DD DISP=SHR,DSN=TWS.INST.PARM(BATCHOPT)
//LTREPORT DD DSN=TWSRES4.TWSC.LTEXT.LIST,DISP=SHR,
//         DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6050)
//EQQMLOG  DD DISP=SHR,DSN=TWS.INST.MLOG
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSMDUMP DD DISP=MOD,DSN=TWS.INST.SYSDUMPB
//EQQDUMP  DD SYSOUT=*
//EQQDMSG  DD SYSOUT=*
//LTOLIN   DD DCB=(RECFM=VB,LRECL=1000,BLKSIZE=6220),
//         SPACE=(CYL,(9,9)),UNIT=3390
//LTOLOUT  DD DCB=(RECFM=VB,LRECL=1000,BLKSIZE=6220),
//         SPACE=(CYL,(9,9)),UNIT=3390
//LTPRIN   DD DCB=(RECFM=FB,LRECL=65,BLKSIZE=4550),
//         SPACE=(4550,(900,900)),UNIT=3390
//LTPROUT  DD DCB=(RECFM=FB,LRECL=65,BLKSIZE=4550),
//         SPACE=(4550,(900,900)),UNIT=3390
//LTOCIN   DD DCB=(RECFM=FB,LRECL=735,BLKSIZE=4410),
//         SPACE=(4410,(900,900)),UNIT=3390
//LTOCOUT  DD DCB=(RECFM=FB,LRECL=735,BLKSIZE=4410),
//         SPACE=(4410,(900,900)),UNIT=3390
//LTOLWK01 DD SPACE=(CYL,(9,9)),UNIT=3390
//LTOLWK02 DD SPACE=(CYL,(9,9)),UNIT=3390
//LTOLWK03 DD SPACE=(CYL,(9,9)),UNIT=3390
//LTPRWK01 DD SPACE=(CYL,(9,9)),UNIT=3390
//LTPRWK02 DD SPACE=(CYL,(9,9)),UNIT=3390
//LTPRWK03 DD SPACE=(CYL,(9,9)),UNIT=3390
//LTOCWK01 DD SPACE=(CYL,(9,9)),UNIT=3390
//LTOCWK02 DD SPACE=(CYL,(9,9)),UNIT=3390
//LTOCWK03 DD SPACE=(CYL,(9,9)),UNIT=3390
//EQQADDS  DD DSN=TWS.INST.TWSC.AD,DISP=SHR
//EQQWSDS  DD DSN=TWS.INST.TWSC.WS,DISP=SHR
//EQQLTDS  DD DSN=TWS.INST.TWSC.LT,DISP=SHR,
//            AMP=('BUFNI=10,BUFND=10')
//EQQLTBKP DD DSN=TWS.INST.TWSC.LB,DISP=SHR
//EQQLDDS  DD DSN=TWS.INST.TWSC.LD,DISP=SHR,
//            AMP=('BUFNI=10,BUFND=10')
//*%OPC SCAN
//*%OPC SETVAR TLTP=(YYMMDD+28CD)
//*
//SYSIN    DD *
&TLTP
//*
//*   Licensed Materials - Property of IBM
//*   5697-WSZ
```

```
//*   (C) Copyright IBM Corp. 1990, 2003 All Rights Reserved.
//*   US Government Users Restricted Rights - Use, duplication
//*   or disclosure restricted by GSA ADP Schedule Contract
//*   with IBM Corp.
//*
//* YYMMDD      DDDD        WHERE
//* YYMMDD                = EXTEND DATE OR BLANK
//*             DDDD      = PLAN EXTENSION IN DAYS
//*                         COUNTING ALL DAYS
//*                         OR BLANK
/*
```

The same thing can be done for the current plan as well, but you probably will want to change the variable to one calendar day or work day.

10.4 Tivoli Workload Scheduler for z/OS variables on the run

This section shows how to update job-scheduling variables in the work flow. Job scheduling uses Tivoli Workload Scheduler for z/OS user variables to pass parameters, qualify filenames, and control work flow. One aspect of user variables that is not obvious is how to update the variables on the fly based on their current value.

10.4.1 How to update Job Scheduling variables within the work flow

Why would you want to do this? You may want to increment a counter every time a certain set of jobs runs so that the output reports have unique numbers. Or, if you need to read a different input file from a set of 20 files each time a certain job runs, you could increment a counter from 1 to 20 and when the counter exceeds 20, set it to 1 again. One customer uses an absolute generation number in their jobs and updates this each business day. You can do this and more with the Tivoli Workload Scheduler for z/OS Control Language (OCL), which provides the ability to update Tivoli Workload Scheduler for z/OS user variables from a batch job.

10.4.2 Tivoli Workload Scheduler for z/OS Control Language (OCL)

Seven components are needed to position properly to use OCL:

EQQOCL The complied OCL REXX code. This is provided with Tivoli Workload Scheduler for z/OS in SEQQMISC. It requires the IBM Compiler Libraries for REXX/370 Version 1.3.0 or later,

Program Number 5695-014. Note that the REXX Alternate Library will not work.

EQQPIFT The program used by OCL to perform UPD and SETUPD functions that change the default value of a user variable. Source code is provided with Tivoli Workload Scheduler for z/OS in SEQQSAMP for COBOL and PL1 compilers respectively in members EQQPIFJC and EQQPIFJV.

EQQYRPRC The JCL proc used to execute EQQOCL. A sample is provided in SEQQSAMP.

EQQYRJCL A job to execute the EQQYRPRC proc and provide the control statements to update JCL variables. This job must be submitted by Tivoli Workload Scheduler for z/OS to retrieve the current JCL variable value. A sample is provided in SEQQSAMP.

EQQYRMSG OCL messages. This is provided in SEQQSAMP.

PIFOPTS PIF options. OCL uses the PIF. You must define these but they are simple.

To customize and position the OCL components:

1. Place EQQOCL, the compiled OCL REXX code, where you want to run it (for example, leave in SEQQMISC or copy to a Tivoli Workload Scheduler for z/OS user REXX library). It must be in the SYSEXEC DD statement concatenation in the EQQYRPRC JCL proc.
2. Compile and link-edit the source for the EQQPIFT program, which is provided in SEQQSAMP in members EQQPIFJC (COBOL) and EQQPIFJV (PL1). Copy the link-edited load module into a Tivoli Workload Scheduler for z/OS user loadlib that is authorized to z/OS and can be accessed by the EQQYRPRC proc from Linklist or a STEPLIB.
3. Copy EQQYRPRC from SEQQSAMP into a user proclib and customize the DD file allocations, Refer to the other steps.
4. Copy EQQYRJCL from SEQQSAMP into the EQQJBLIB Joblib PDS and customize the job card and DD statements so that it can be run by the Tivoli Workload Scheduler for z/OS Controller. Use this as a reference to create and customize a separate job for each separate set of JCL variables that need to be updated.
5. Copy EQQYRPRM into the Tivoli Workload Scheduler for z/OS parmlib and customize. Specify TSOCMD (YES), and specify the appropriate Controller and Tracker subsystem names; for example, SUBSYS (TWSC) and OPCTRK (TWST).

6. Place EQQYRMSG where you want it (for example, leave in SEQQSAMP or copy to a Tivoli Workload Scheduler for z/OS user message library). The OCMLMIB DD statement in EQQYPRC must point to this.
7. Create a PIF options member in your Tivoli Workload Scheduler for z/OS parmlib, such as PIFOPTS. The options needed are few, for example:
   - INIT CWBASE(00)
   - HIGHDATE (711231)

   The EQQYPARM DD statement in EQQYRPRC must point to this.
8. Test.
   a. Create a user variable table and insert at least one numeric variable into the table and set the initial value.
   b. Create a job in Joblib based on EQQYRJCL, and customize it to update your variable.
   c. Add the job to the database using primary men 1.4 or 1.8.
   d. Add the job to the current plan.
   e. Check the results.

To define a user variable (see also 10.3.1, “Setting up a table” on page 250):

1. Using the Tivoli Workload Scheduler for z/OS dialogs from anywhere in Tivoli Workload Scheduler, choose =1.9.2 to create a new JCL variable table or modify.
2. Create a variable table called OCLVARS.

If the table does not yet exist, the next panel enables you to create a new table into which you can insert variables and set the initial value. If a table already exists, you will be presented with the table name in a list, Select the table using M for modify, and you will be able to insert more variables into the table or select existing variables for update.

10.4.3 Tivoli Workload Scheduler for z/OS OCL examples

Example 10-28 shows a job customized from the EQQYRJCL sample to increment by one the variable OPCVAR1 in the JCL variable table OPCVARS.

Example 10-28 EQQYRJCL sample to increment by 1

```
//WSCYRJCL JOB
//MYJCLLIB JCLLIB ORDER=OPCESA.V8R2M0GA.USRPROC
//*%OPC SCAN
//*%OPC TABLE NAME=(OCLVARS)
//EQQOCL EXEC EQQYRPRC
```

```
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=FB,LRECL=133,BLKSIZE=1330)
//SYSTSPRT DD SYSOUT=*
//EQQOCL.SYSIN DD *
INIT VARTAB(OCLVARS) SUBSYS(TWSC)
SETUPD OCLVAR1 = &OCLVAR1 + 1
/*
```

Example 10-29 uses REXX code to increment the variable to a maximum value and then start over again at 1.

Example 10-29 REXX code to increment the variable

```
//WSCYRJC2 JOB
//*%OPC SCAN
//*%OPC TABLE NAME=(OCLVARS)
//EQQOCL EXEC EQQYRPRC
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=FB,LRECL=133,BLKSIZE=1330)
//SYSTSPRT DD SYSOUT=*
//EQQOCL.SYSIN DD *
INIT VARTAB(OCLVARS) SUBSYS(TWSC)
SETUPD OCLVAR2 = @UP('&OCLVAR2',1,20)
```

Example 10-30 is a customized EQQYRPRC proc from MYJCLLIB.

Example 10-30 Customized EQQYRPRC

```
//EQQOCL EXEC PGM=IKJEFT01,PARM='EQQOCL'
//STEPLIB DD DISP=SHR,DSN=OPCESA.V8R2MOGA.SEQQLMD0
// DD DISP=SHR,DSN=SYS1.LEMVS.SCEERUN
//OCLLOG DD DISP=MOD,DSN=OPCESA.V8R2MOGA.OCL.LOG,
// DCB=(RECFM=FB,LRECL=133,BLKSIZE=1330)
//OCLPARM DD DISP=SHR,DSN=OPCESA.V8R2MOGA.PARMLIB(EQQYRPRM) //OCLMLIB
DD DISP=SHR,DSN=OPCESA.V8R2MOGA.SEQQSAMP(EQQYRMSG) //EQQMLIB DD
DISP=SHR,DSN=OPCESA.V8R2MOGA.USER.SEQQMSG0
// DD DISP=SHR,DSN=OPCESA.V8R2MOGA.SEQQMSG0
//EQQYPARM DD DISP=SHR,DSN=OPCESA.V8R2MOGA.PARMLIB(PIFOPTS) //SYSEXEC
DD DISP=SHR,DSN=OPCESA.V8R2M0.USREXEC
// DD DISP=SHR,DSN=OPCESA.V8R2MOGA.SEQQMISC
//CARDIN DD UNIT=SYSDA,SPACE=(TRK,(20,200)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=FB,LRECL=133,BLKSIZE=1330) //SYSTSPRT
DD SYSOUT=*
//SYSTSIN DD
DUMMY
```

11

Audit Report facility

This chapter discusses basic use of the Tivoli Workload Scheduler for z/OS Audit Report function. The information provided will assist with investigating particular problems that have occurred within the schedule. In effect, the Audit Report will help pinpoint problem areas that must be resolved.

This chapter covers the following topics:

- What is the audit facility?
- Invoking the Audit Report interactively
- Submitting from the dialog a batch job
- Submitting an outside batch job

11.1 What is the audit facility?

The audit facility performs an examination of the Tivoli Workload Scheduler for z/OS scheduling job-tracking or track-log data sets. The audit facility is located in the Optional Function section in the primary menu panel of Tivoli Workload Scheduler for z/OS (Figure 11-1).

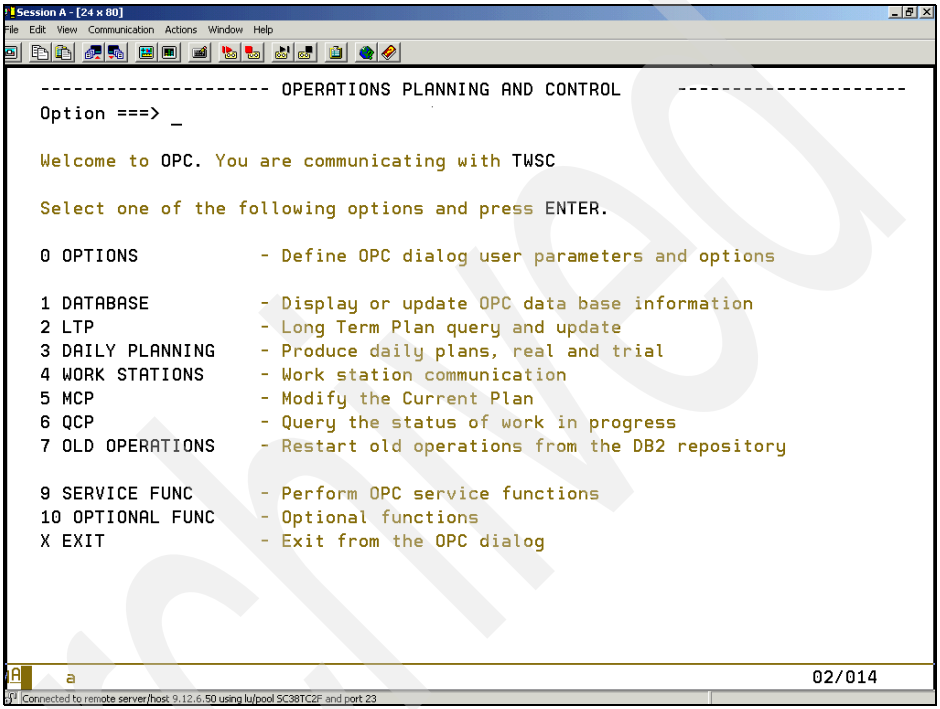


Figure 11-1 Operations Planning and Control main menu

When you select option 10, a small menu appears (Figure 11-2) with the option of invoking the audit function interactively via option 10.1. You can also submit a batch job by choosing 10.2.

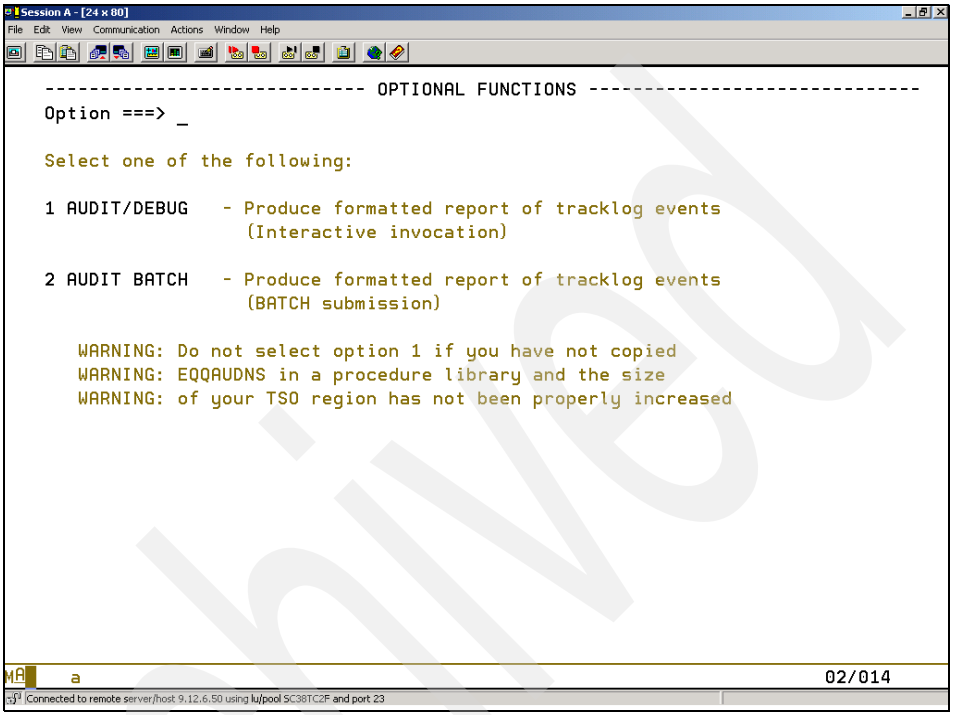


Figure 11-2 Tivoli Workload Scheduler for z/OS Optional Functions menu

11.2 Invoking the Audit Report interactively

As indicated in Figure 11-2, select option 1 from the menu AUDIT/DEBUG. You can access this option from anywhere in Tivoli Workload Scheduler for z/OS by typing `=10.1` on the command/option line.

On the Audit Facility panel, you must enter either JTX or TRL:

JTX — Contents of the current Job Track Logs; that is, anything that was performed past the initial run of the current plan.

TRL — Contents of the archive of the Job Track Logs (data prior to the last current plan run).

Next, enter the name of the text you wish to search for on the search-string line. For example, you can enter the jobname, application name, workstation name, or

user ID. You can also use wildcards to perform more global searches, such as using an asterisk (*) in various formats such as:

- TESTJOB
- TEST*
- *JOB
- CPU1
- CPU*
- C*
- USER1
- USER*

Using these wildcard examples provides either general or vast amounts of information. In most cases, it is best to be more specific unless you are unsure of the actual application name, jobname, username, or other criterion.

You can also narrow your search by entering the start date and time that you wish to start from, and then enter the end date and time of the search. This is optional and you can leave the date/time blank, but it is good practice to use the date/time to limit the size of the report and the processing time.

The Audit Report shows:

- The actual start and completion or abend of an operation (including restarts).
- User who made changes to an operation or application in the current plan and the Tivoli Workload Scheduler for z/OS database.
- How did the operation actually complete or abend?
- A reason for the rerun if it was coded while performing the Restart and Cleanup.
- Any JCL changes that were made during the Restart and Cleanup.

11.3 Submitting from the dialog a batch job

In this case, you can issue a batch job to produce an extended Audit Report. As with 10.1 and 10.2 of the dialog, both methods are useful when there is an urgency to create reports from the job-tracking or track-log data sets.The basic use of this report is a matter of saving time over finding answers to questions without having to take a lot of time examining the input records with the use of the mappings that are in *IBM Tivoli Workload Scheduler for z/OS Diagnosis Guide and Reference Version 8.2*, SC32-1261.

Figure 11-3 shows how to generate a batch job from option 10.2.

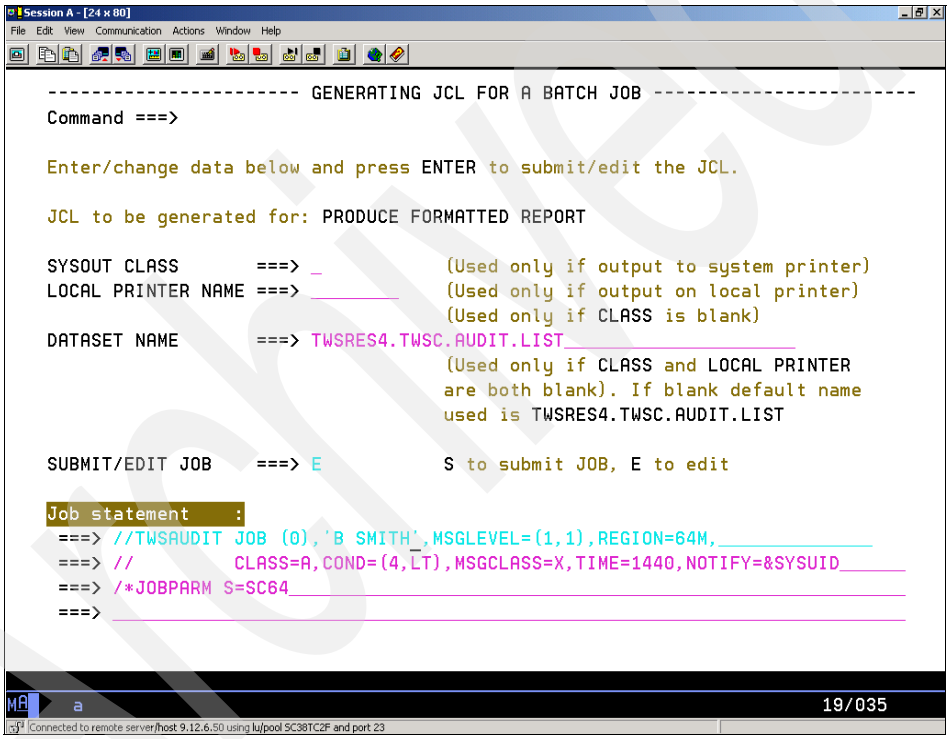


Figure 11-3 Audit Reporting Facility Generating JCL for a Batch Job - 10.2

Here you need a valid job card to submit or edit the job. You can override the default data set name that the report will go to by typing the name in the Dataset Name field. You have two options: submit or edit the job. If you choose submit, the job will run and default to JTX for the current planning period. You can also edit the JCL and make changes or define the search within the JCL. (See Figure 11-4).

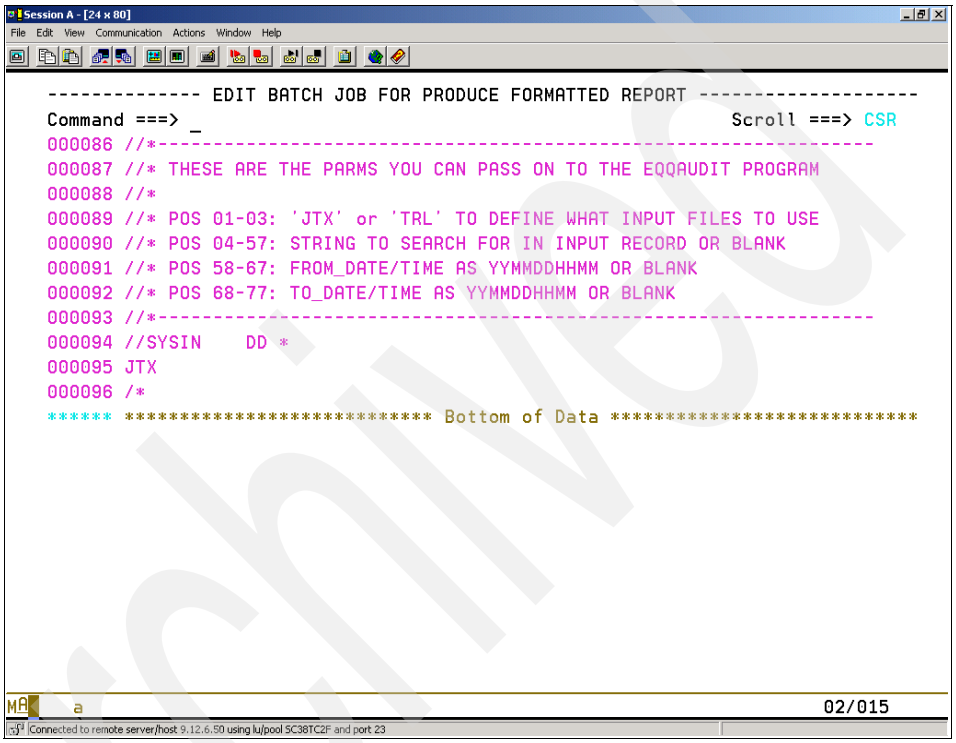


Figure 11-4 Sample JCL for generating a batch job

As Figure 11-4 shows, you have the option to make any changes or updates to assist you in creating the report. You can change the JTX to TRL for a report based on the previous planning period. You can also insert the search string data and the from/to date/time to refine your search. Otherwise, leaving the TRL or JTX as is produces a customized report for the entire current planning period (JTX) or the previous planning period (TRL).

11.4 Submitting an outside batch job

The sample library member EQQAUDIB contains a job that is customized at installation time and that you can submit outside the dialog to start the audit function when either of the two simpler methods described in the previous sections cannot be used.

This alternate way is useful for a planned utilization; many installations have a need to create and store audit trails for a set period of time. In this case, Tivoli Workload Scheduler for z/OS audit job (copied from the EQQAUDIB CUSTOMIZED) can be defined to run automatically after every plan EXTEND or REPLAN or before the EXTEND or REPLAN, using the data set referenced by EQQTROUT as input. Even if you are not required to create an audit trail regularly, the generated report can provide quick answers in determining who in your organization requested a function that caused some business application to fail, or to trace the processing of a job that failed and was rerun many times. If your AUDIT initialization statement specifies all data for JS update requests, you can use the Audit Report to compare against the master JCL to determine exactly which JCL statements were changed.

Example 11-1 shows the header page for the Tivoli Workload Scheduler for z/OS Audit Report. The input source here is TRL, though it could have been JTX based on the choice that was made from the panel or from the batch job. The search-string and date/time are chosen from the panel or batch job, and can be left blank.

Example 11-1 Tivoli Workload Scheduler for z/OS Audit header page

```
DATE/TIME OF THIS RUN: 940805 01.01
*************************************************************
* SAMPLE OPC/ESA AUDIT/DEBUG REPORT *
* P R O G R A M P A R A M E T E R S *
*************************************************************
* INPUT SOURCE : TRL *
* SEARCH-STRING : *
* START DATE : *
* START TIME : *
* END DATE : *
* END TIME : *
*************************************************************
************************************************
* LINES INSERTED BY PROGRAM ARE MARKED '=====>' *
* UNKNOWN FIELD VALUES ARE PRINTED AS '?' *
* SUMMARY OF SELECTED RECORDS PRINTED ON LAST PAGE *
************************************************
```

Example 11-2 shows a sample Audit Report. Several records that start with either IJ, A1, A2, or A3 (in boldface) show the different processing steps that the job goes through. IJ indicates that the job has been submitted for processing, and the submit JCL records are produced when Tivoli Workload Scheduler for z/OS submits the job. A1 indicates that the job card has been read. A2 indicates the start of the job. A3 indicates either the successful completion of the job or the abend and reason for the job's failure. When processing this report via batch or without a search string, the output can become quite extensive. The best approach to a search is to use this command to find specific information in the audit file: **X ALL;F 'searchword' ALL**. The 'searchword' can be a user ID, jobname, application name, abend type, time, or date, for example.

Example 11-2 Audit Report sample

```
=============> NOW READING FROM EQQTROUT
08/04 13.55.00 CP UPDT BY XMAWS3 MCP MODIFY 0.10SEC APPL: TB2INVUP IA: 940802 0900
PRTY: 5
- OPNO: 15 TYPE: EX-COMMAND ISSUED
08/04 13.55.01 25 SCHD BY OPC JOBNAME: TB2INVUP AD: TB2INVUP OCC IA: 9408020900
TOKEN:
08/04 13.55.01 CP UPDT BY OPC_WSA OP. CPU1_15 IN TB2INVUP IS SET TO S JOBNAME:
TB2INVUP
08/04 13.55.03 29 PROCESSED IJ-SUBMIT JCL AD/IA: TB2INVUP 9408020900
TB2INVUP(JOB01542)
08/04 13.55.03 29 PROCESSED A1-JOB CARD READ TB2INVUP(JOB01542) AT: 15.55.02.46
08/04 13.55.04 29 PROCESSED A2-JOB START TB2INVUP(JOB01542) AT: 15.55.03.41 ON NODE:
LDGMVS1
08/04 13.55.05 29 PROCESSED A3-STEP END TB2INVUP(JOB01542) AT: 15.55.05.42 PRSTEP
CODE: I0
08/04 13.55.05 29 PROCESSED A3-JOB COMPLETE TB2INVUP(JOB01542) AT: 15.55.05.46 CODE:
0
08/04 13.55.07 CP UPDT BY OPC_JT OP. CPU1_15 IN TB2INVUP IS SET TO E JOBNAME:
TB2INVUP ERROR CODE: JCL
08/04 13.55.07 29 PROCESSED A3-JOB TERMINATE TB2INVUP(JOB01542) AT: 15.55.06.17
08/04 13.55.10 26 AUTO RECOVERY OF: TB2INVUP OCC INP. ARR: 9408020900 RECOVERY DONE
08/04 13.55.24 29 PROCESSED JI-LOG RETR INIT AD/IA: TB2INVUP OP: 015 USR: XMAWS3
08/04 13.55.24 29 PROCESSED JO-LOG RETR STARTED AD/IA: TB2INVUP 9408020900 OP: 015
EXITNAME:
08/04 13.55.26 29 PROCESSED NF-LOG RETR ENDED AD/IA: TB2INVUP 9408020900 OP: 015 RES:
08/04 13.56.27 JS UPDT BY XMAWS3 KEY: TB2INVUP 9408020900 OPNO: 15
08/04 13.56.43 29 PROCESSED CI-CAT.MGMT INIT AD/IA: TB2INVUP OP: 015/
08/04 13.56.43 29 PROCESSED CO-CAT.MGMT STARTED AD/IA: TB2INVUP 9408020900 OP: 015 #
D
08/04 13.56.44 CP UPDT BY XMAWS3 MCP RERUN 0.15SEC APPL: TB2INVUP IA: 940802 0900
PRTY: 5 RE
```

```
RESTART OF: TB2INVUP CONFIRMED ON PANEL EQQMERTP
ERROR CODE: JCL USERDATA:
REASON : do it
- OPNO: 15 TYPE: JOB STATUS NEW OP. STATUS: W
08/04 13.56.46 34 JOB TB2INVUP(JOB01542) NODE: LDG1 APPL: TB2INVUP INP.ARR: 940
- DELETED :EID.EID4R2.J015.CATTEST //DD1 PROCSTEP:
08/04 13.56.46 29 PROCESSED C1-CAT.MGMT ACTIONS AD/IA: TB2INVUP 9408020900 OP:
015/STEP1 RES:
08/04 13.56.46 29 PROCESSED C2-CAT.MGMT ENDED AD/IA: TB2INVUP 9408020900 OP: 015 # D
08/04 13.57.08 CP UPDT BY XMAWS3 MCP MODIFY 0.15SEC APPL: TB2INVUP IA: 940802 0900
PRTY: 5
- OPNO: 15 TYPE: EX-COMMAND ISSUED
=============> EOF REACHED ON EQQTROUT
```

In the example, the boldface shows the process of job TB2INVUP as it has ended in error in Tivoli Workload Scheduler. You can see the process where Catalog Management is invoked and the job is rerun in the current plan along with the reason for restart.

Example 11-3 from the Audit Report is an out-of-sequence event (or essentially an out-of-sequence event). A suspended event occurs when Tivoli Workload Scheduler for z/OS is running in either a JES2 shared pool complex or a JES3 Global/Local complex. A job may be processing on several different LPARs and be going through different phases, with each phase being processed, then sent to the Tracker on the LPAR it is running on. This information is also sent to the Controller in multiple different paths. Based on this it could arrive “out of sequence.” A job end event could come before the actual job start event, for example. So when this happens, Tivoli Workload Scheduler for z/OS suspends the “early” event and put it in a Suspend queue.

Example 11-3 Suspended event

```
04.00.17 29 PROCESSED IJ-SUBMIT JCL     AD/IA: MQBACKUPZPA     030319180
04.00.19 29 PROCESSED A1-JOB CARD READ  MQBBCKR6(JOB19161)  AT: 04.00.19
04.00.19 29 PROCESSED A2-JOB START      MQBBCKR6(JOB19161)  AT: 04.00.19
04.19.16 29 SUSPENDED A5-JOB PURGE      MQBBCKR6(JOB19161)  AT: 04.19.16
04.24.19 29 PROCESSED A5-JOB PURGE      MQBBCKR6(JOB19161)  AT: 04.19.16
05.07.16 29 DISCARDED A3-JOB TERMINATE  MQBBCKR6(JOB19161)  AT: 04.19.11
```

If a suspended event stays in the queue for more than five minutes, it will be discarded and Tivoli Workload Scheduler for z/OS will make the best judgement of the jobs status based on the actual information received. All in all, an occasional suspended record in the Audit Report is not a cause for concern, especially because the suspended condition is eventually resolved.

If there is a tremendous amount of suspended events, that will be cause for concern. This could be a performance or communication problem with the Tracker (or Trackers), and this should be looked into immediately. If an event record is lost, all successor events for that job will be suspended and soon discarded, leaving that job in its last valid status until it is eventually purged from the system. The JES2 HASP250 message is issued, and a Tivoli Workload Scheduler for z/OS A5 event record is created. When the Controller gets that event for a job that is not in a completed or error status, it realizes that there is a serious problem and thus the job is set to CAN status. The only proper way for a job to go right to started to purge status is if it were canceled.

The end of the Audit Report has a summary of the event types processed (Example 11-4), its corresponding number (for example, 29 for Automatic Operations), the number of records read, and the events selected. This can be used for customer statistics based on the prior days' production.

Example 11-4 Audit Report sample

```
DATE/TIME OF THIS RUN: 940805 01.01
***********************************************
*                              *RECORDS * EVENTS *
* E V E N T T Y P E         NO * READ   *SELECTED*
***********************************************
* DAILY PLAN STATUS RECORD   01 * 000001 * 000000 *
* DAILY PLAN WORK STATIONS   02 * 000059 * 000000 *
* DAILY PLAN OCC/OPER. RCDS  03 * 013266 * 000000 *
* JOB TRACKING START         20 * 000005 * 000005 *
* MANUAL OPERATIONS          23 * 000328 * 000328 *
* MODIFY CURRENT PLAN        24 * 000013 * 000013 *
* OPERATION SCHEDULED        25 * 000006 * 000006 *
* AUTO RECOVERY              26 * 000006 * 000006 *
* FEEDBACK DATA              28 * 000112 * 000112 *
* AUTOMATIC OPERATIONS       29 * 000226 * 000226 *
* DATA BASE UPDATE           32 * 000016 * 000016 *
* CATALOG MANAGEMENT EVENT   34 * 000003 * 000003 *
* BACKUP LOG RECORD          36 * 000007 * 000007 *
* DATA LOG RECORD               * 000007 * 000007 *
***********************************************
*************************************************************
* MCP PERFORMANCE       * NO OF   * E L A P S E D T I M E *
* TYPE OF UPDATE        * UPDATES * M I N * M A X * A V G *
*************************************************************
* MCP RERUN             * 000006  * 0.03  * 0.78  * 0.22  *
* MCP MODIFY            * 000007  * 0.01  * 0.16  * 0.09  *
*************************************************************
LONGEST MCP-EVENTS:
```

```
08/04 13.41.54 CP UPDT BY XMAWS3 MCP RERUN 0.78SEC APPL: TB6INTRP IA: 940802 1400
PRTY: 5
08/04 13.59.52 CP UPDT BY XMAWS3 MCP RERUN 0.16SEC APPL: TB2INVUP IA: 940802 0900
PRTY: 5
08/04 13.49.16 CP UPDT BY XMAWS3 MCP MODIFY 0.16SEC APPL: TB2INVUP IA: 940802 0900
PRTY: 5
***** END OF REPORT *****
```

Audit Report JCL

Example 11-5 is a sample copy of the JCL used for the Audit Report. It is a good practice to set up an Audit job on at least a daily basis prior to the next run of the Long-term and current plan. This provides the prior days' complete audit.

Example 11-5 Audit Report JCL

```
//TWSAUDIT JOB (0),'B SMITH',MSGLEVEL=(1,1),REGION=64M,
//         CLASS=A,COND=(4,LT),MSGCLASS=X,TIME=1440,NOTIFY=&SYSUID
/*JOBPARM S=SC64
//DELETE   EXEC PGM=IEFBR14
//OLDLIST  DD DSN=TWSRES4.TWSC.AUDIT.LIST,
//         UNIT=3390,SPACE=(TRK,0),DISP=(MOD,DELETE)
//ALLOC    EXEC PGM=IEFBR14
//NEWLIST  DD DSN=TWSRES4.TWSC.AUDIT.LIST,
//         UNIT=3390,DISP=(,CATLG),SPACE=(CYL,(9,9),RLSE),
//         DCB=(RECFM=FBA,LRECL=121,BLKSIZE=12100)
//*  This program will take input from either the JTARC/JTx-files (for
//*  reporting in real time) or from the //EQQTROUT-file of the daily
//*  plan extend/replan programs (for after-the-fact reporting).
//*
//*  If not excluded by the user-defined filters, each tracklog-event
//*  will generate one formatted report-line (with some exceptions in
//*  which more lines are created). The program will also:
//*    - Print all JCL-lines if AMOUNT(DATA) specified for FILE(JS)
//*        (If PASSWORD= is encountered, the password will be blanked
//*         out and not appear in the listing)
//*    - Print all variable values of a job if AMOUNT(DATA) was
//*        specified for FILE(VAR)
//*    - Print all origin dates of a period if AMOUNT(DATA) was
//*        specified for FILE(PER)
//*    - Print all specific dates of a calendar if AMOUNT(DATA) was
//*        specified for FILE(CAL)
//*  You have to specify AMOUNT(DATA) for FILE(LTP) to have for
//*    example deleted occurrences identified in a meaningful way.
//*  An MCP-request will be broken down into sub-transactions and
```

```
//*    each sub-transaction listed in connection to the originating
//*    request:
//*    - All WS intervals will be printed if availability of a WS is
//*        changed
//*    - All contained group occurrences will be listed for a 'group'
//*        MCP request
//*  If you specify a search-string to the program it will select
//*  events with the specified string in it somewhere. That MAY mean
//*  that you will not the see the string in the report-line itself
//*  as the line may have been too 'busy' to also have room for your
//*  string value, whatever it may be.
//*********************************************************************
//* EXTRACT AND FORMAT   OPC      JOB TRACKING EVENTS                 *
//*********************************************************************
//AUDIT    EXEC PGM=EQQBATCH,PARM='EQQAUDIT',REGION=4096K
//STEPLIB  DD DISP=SHR,DSN=EQQ.SEQQLMDO
//EQQMLIB  DD DISP=SHR,DSN=EQQ.SEQQMSGO
//EQQPARM  DD DISP=SHR,DSN=TWS.INST.PARM(BATCHOPT)
//EQQMLOG  DD DISP=SHR,DSN=TWS.INST.MLOG
//SYSUDUMP DD SYSOUT=*
//EQQDUMP  DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*,
//         DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6050)
//*--------------------------------------------------------------------
//* FILE BELOW IS CREATED IN DAILY PLANNING BATCH AND USED IF INPUT
//* OPTION IS 'TRL'
//EQQTROUT DD DISP=SHR,DSN=TWS.INST.TRACKLOG
//*--------------------------------------------------------------------
//*
//*--------------------------------------------------------------------
//* FILES BELOW ARE THOSE SPECIFIED IN STC-JCL FOR THE OPC
//* CONTROLLER SUBSYSTEM AND USED IF INPUT OPTION IS 'JTX'.
//EQQCKPT  DD DISP=SHR,DSN=TWS.INST.TWSC.CKPT
//EQQJTARC DD DISP=SHR,DSN=TWS.INST.TWSC.JTARC
//EQQJTO1  DD DISP=SHR,DSN=TWS.INST.TWSC.JT1
//EQQJTO2  DD DISP=SHR,DSN=TWS.INST.TWSC.JT2
//EQQJTO3  DD DISP=SHR,DSN=TWS.INST.TWSC.JT3
//EQQJTO4  DD DISP=SHR,DSN=TWS.INST.TWSC.JT4
//EQQJTO5  DD DISP=SHR,DSN=TWS.INST.TWSC.JT5
//*--------------------------------------------------------------------
//*
//*--------------------------------------------------------------------
//* FILE BELOW IS THE MLOG WRITTEN TO BY THE CONTROLLER SUBSYSTEM.
//* FOR PERFORMANCE AND INTEGRITY IT IS RECOMMENDED TO LEAVE IT
```

```
//* DUMMY. IF YOU REALLY WANT TO HAVE THE OUTPUT INCLUDING MLOG,
//* YOU CAN USE THE REAL NAME, RUNNING THE EQQAUDIB SAMPLE WHEN THE
//* SUBSYSTEM IS STOPPED OR USING A COPY OF THE LIVING MLOG.
//LIVEMLOG DD DUMMY
//*--------------------------------------------------------------------
//*
//*--------------------------------------------------------------------
//* FILE BELOW IS WHERE THE REPORT IS WRITTEN.
//AUDITPRT DD DSN=TWSRES4.TWSC.AUDIT.LIST,DISP=SHR,
//         DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6050)
//*--------------------------------------------------------------------
//*--------------------------------------------------------------------
//* THESE ARE THE PARMS YOU CAN PASS ON TO THE EQQAUDIT PROGRAM
//*
//* POS 01-03: 'JTX' or 'TRL' TO DEFINE WHAT INPUT FILES TO USE
//* POS 04-57: STRING TO SEARCH FOR IN INPUT RECORD OR BLANK
//* POS 58-67: FROM_DATE/TIME AS YYMMDDHHMM OR BLANK
//* POS 68-77: TO_DATE/TIME AS YYMMDDHHMM OR BLANK
//*--------------------------------------------------------------------
//SYSIN    DD *
JTX
/*
```

12

Using Tivoli Workload Scheduler for z/OS effectively

In this chapter, we discuss considerations to maximize the Tivoli Workload Scheduler for z/OS output and optimize the job submission and status feedback.

The customizations and best practices covered in this chapter include the following topics:

- Prioritizing the batch flows
- Designing your batch network
- Moving JCL into the JS VSAM files
- Recommendations

12.1 Prioritizing the batch flows

This section describes how Tivoli Workload Scheduler for z/OS prioritizes the batch and how to exploit this process.

It specifically looks at the use of correct duration and deadline times, and how these can be simply maintained.

12.1.1 Why do you need this?

So why do you need this? In other words, why do you care about correct duration and deadline times and why do you need to exploit the Tivoli Workload Scheduler for z/OS batch prioritization process?

Tivoli Workload Scheduler for z/OS will run all the jobs for you, usually within your batch window, but what about when something goes wrong—what impact does it have, and who even knows whether it will affect an online service the next day?

The goals should be to:

- Complete the batch within time scales
- Prioritize on the critical path
- Understand the impact of errors

Today, the problems in reaching these goals are numerous. It seems like only a few years ago that every operator knew every job in the system, what it did, and the impact if it failed. Of course, there were only a few hundred jobs a night then. These days, the number of batch jobs run into thousands per night, and only a few are well known. Also, their impact on the overall batch might not be what the operators remember any more. So, how can they keep up, and how can they make educated decisions about late running and failed jobs?

The solution is not magic. You just give Tivoli Workload Scheduler for z/OS some correct information and it will build the correct priorities for every job that is relevant to every other scheduled job in the current plan.

Using any other in-house designed methods of identifying important work (for example, a high-priority workstation), will soon lose any meaning without regular maintenance, and Tivoli Workload Scheduler for z/OS will not consider these methods when building its plan or choosing which job to run next.

You have to use the planning function of Tivoli Workload Scheduler for z/OS and help it to build the correct priority for every scheduled job based on its criticality to the overall delivery of service.

12.1.2 Latest start time

Although each application has a mandatory priority field, this rarely differentiates one operation priority from another.

The workstation analyzer submits the job from the ready queue that it deems to be the most urgent and it does this by comparing the relative “latest start times” of all ready operations.

So what is a latest start time (sometimes called the latest out time) and where does it come from?

Each time the current plan is extended or replanned, Tivoli Workload Scheduler for z/OS calculates the time by which *every* operation in the current plan must start in order for all its successors to meet any deadlines that have been specified. This calculated time is the latest start time.

12.1.3 Latest start time: calculation

To give you an idea of how Tivoli Workload Scheduler for z/OS calculates the latest start time, consider the example in Figure 12-1 on page 288, which shows a network of jobs.

Assume that each job takes 30 minutes. JOBTONEK, JOBTWOP, and JOBTREY (all shown in bold) should complete before their relative onlines can be made available. Each has an operation deadline time of 06:30 defined. The application they belong to has a deadline time of 09:00.

With this information, Tivoli Workload Scheduler for z/OS can calculate the latest start time of every operation in the network. It can also determine which path through the network is the critical path.

So, for JOBTWOP to complete at 06:30, it must start at 06:00 (06:30 minus 30 minutes). Both of its immediate predecessors must start at 05:30 (06:00 minus 30 minutes). Their immediate predecessors must start at 05:00, and so on, back up the chains to BUPJOB1, which must start by 01:00.

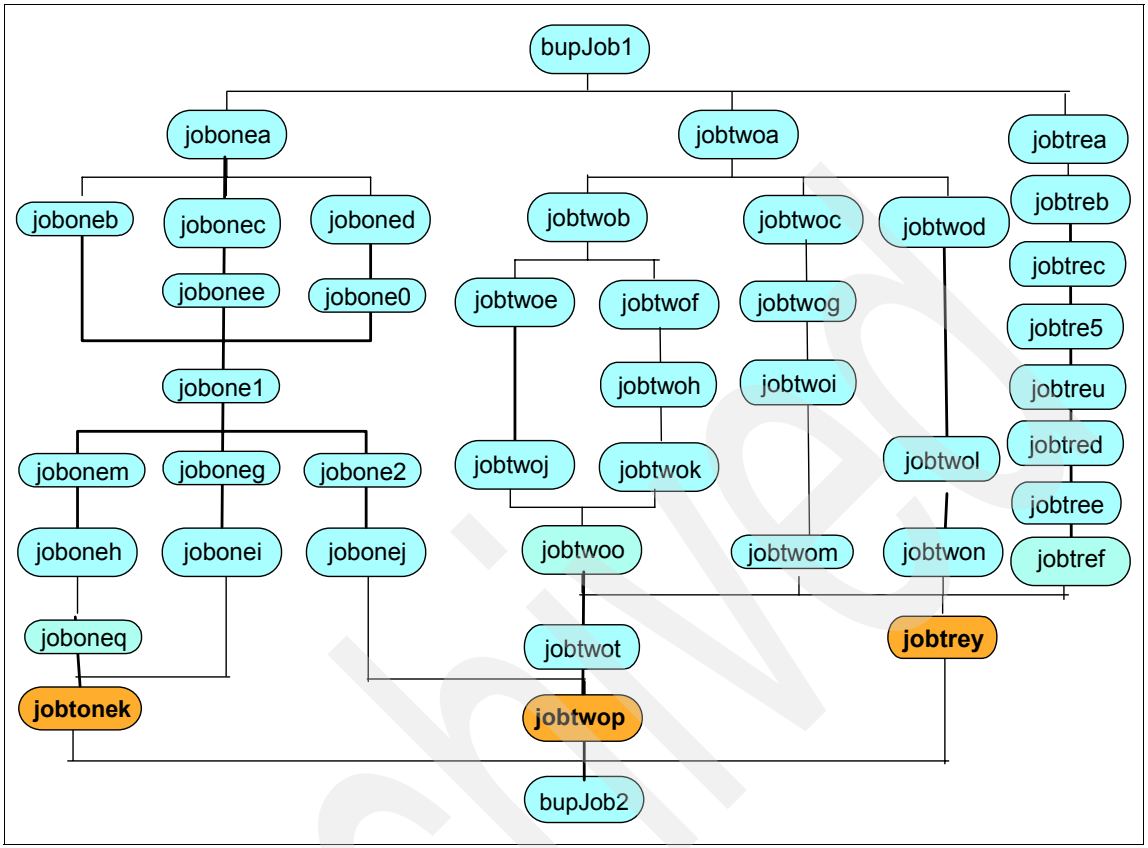


Figure 12-1 Network of jobs

When calculating the latest start times for each job, Tivoli Workload Scheduler for z/OS is, in effect, using the “latest start time” of its successor job in lieu of a deadline time. So if it encounters a more urgent deadline time on an operation in the chain, that deadline will be used instead.

Consider our example in Figure 12-1. The path that takes the longest elapsed time, the critical path, from BUPJOB1 is down the right side, where the number of jobs between it and one of the online startup jobs is greatest. But, what if JOBONE1 produces a file that must be sent to an external company by 02:00? Calculating back up the chain from JOBONEK (06:30 deadline), we have a calculated latest start time on JOBONEM of 04:30. This is not as urgent as 02:00, so JOBONE1 will use its 02:00 deadline time instead and get a latest start time of 01:30. This will affect the whole predecessor chain, so now BUPJOB1 has a latest start time of 23:30 the previous day.

12.1.4 Latest start time: maintaining

When determining which job to submit next, the Workstation Analyzer takes many things into consideration. This might not be important where there are no restrictions on your system, but normally, there are conflicts for resources, system initiators, and so on. All these system restrictions are probably reflected in the special resources and parallel servers defined to Tivoli Workload Scheduler for z/OS.

Then, it becomes important that the next submission is a considered one. Every day, the workload is slightly different from any other previous workload. To maintain the kind of relative priority latest start time provides would require considerable analysis and effort.

Because Tivoli Workload Scheduler for z/OS uses deadline times and operation durations to calculate the latest start times, it makes sense to keep these as accurate as possible.

For durations, you should determine, or review, your policy for using the *limit of feedback* and *smoothing* algorithms. Remember, if you are using them, the global values used in the initialization statements also govern the issuing of “late” and “duration” alerts. Of course, a by-product of this process is the improvement in the accuracy of these alerts.

The daily plan EXTEND job also reports on *missed feedback*, which will enable you to manually correct any duration times that cannot be corrected automatically by Tivoli Workload Scheduler for z/OS.

The other factor to be considered is the deadline times. This is a manual exercise. Investigation into those elements of your schedule that really must be finished by a specified time is unlikely to be a simple one. It might include identifying all those jobs that make your services available, contractual delivery times with external suppliers, banks, tax offices, and so on. In addition, you might have deadline times on jobs that really do not need them.

12.1.5 Latest start time: extra uses

A another way to make this process useful is to use a dummy (non-reporting) workstation to define the deadline times against, rather than the actual jobs. This makes it very easy for everyone to see where the deadlines are.

When investigating why a late alert is received it helps to know which deadline is being compromised. When a job has failed, a quick review of successors will show the relative importance of this failure. Operator instructions or other job documentation can then be raised for these dummy deadline operations that could advise what processes can be modified if the deadline is in jeopardy.

In large installations, this type of easy information provision replaces the need for operators to know the entire batch schedule. The numbers of jobs involved and the rates of change make the accuracy of this old knowledge sporadic at best. Using this process immediately integrates new batch processing into the prioritization.

Each operation has a relative priority in the latest start time, so this can be used to sort the error queue, such that the most urgent failure is presented at the top of the list. Now even the newest operator will know which job must be fixed first.

For warnings of problems in the batch, late and duration alerting could be switched on. After this data has been entered and the plans are relatively accurate, these should be only issued for real situations.

In addition, it becomes easy to create a monitoring job that can run as part of the batch that just compares the current time it is running with its latest start time (it is available as a Tivoli Workload Scheduler for z/OS variable) and issues a warning message if the two times are closer than is prudent.

12.1.6 Earliest start time

It is worthwhile to make an effort to change the input arrival time of each *first operation* in the batch (say, the job that closes a database) to a realistic value. (You do not have to make it time-dependent to do this; only specify a specific time for the operations.) Tivoli Workload Scheduler for z/OS will calculate the earliest time each job can start, using the operation's duration time for its calculation. Checking this earliest start time against reality enables you to see how accurate the plans are. (They will require some cleaning up). However, this calculation and the ability to build new work into Tivoli Workload Scheduler for z/OS long before its live date will enable you to run trial plans to determine the effect of the new batch on the existing workload and critical paths.

A common problem with this calculation is caused by Special Resources that are normally unavailable; they might or might not have been defined in the special resource database. However, if an operation wants to use it and it is unavailable, the plan will see this for eight days into the future (the Special Resource planning horizon) and not be able to schedule a planned start time until then. This is an easy situation to circumvent. Simply define the special resource in the database, or amend its current definition, so that it is used only for *control*, and not for planning or for both planning and control. Now, the special resource will be ignored when calculating the planned start time.

12.1.7 Balancing system resources

There are many system resources for which batch jobs contend. Including these in your operation definitions as special resources will better enable Tivoli Workload Scheduler for z/OS to calculate which job should be submitted next. If it does not know that some insignificant job uses all of the IMS batch message processing (BMP) that the very important job wants, it will submit them both, and the critical job always ends up failing.

Parallel servers or a Special Resource should also be used to match the number of system initiators available for batch (or the optimum number of batch jobs that should run at any one time). This is because each time a job ends it might be releasing a critical path job, which, on entry to JES, just sits waiting behind lower-priority jobs for an initiator, or gets swapped out by Workload Manager (WLM) because it does not have the same prioritization process as IBM Tivoli Workload Scheduler for z/OS.

12.1.8 Workload Manager integration

Many installations define what they think are their critical jobs to a higher WLM service class; however, WLM only knows about the jobs that are actually in the system now.

Tivoli Workload Scheduler for z/OS knows about the jobs that still have to run. It also knows when a job's priority changes, maybe due to an earlier failure.

By creating a *very hot batch service class* for Tivoli Workload Scheduler for z/OS to use, it is possible for it to move a job into this hot service class if the job starts to overrun or is late.

12.1.9 Input arrival time

Although not directly related to the performance, input arrival time is a very important, and sometimes confused, concept in Tivoli Workload Scheduler for z/OS. For this reason, we want to explain this concept in more detail.

Let us first clear up a common misunderstanding: *Input arrival time has no relation with time dependency. It does not indicate when the job stream (application) or the jobs (operation) in the job stream are allowed or expected to run.* If you need to give a time restriction for a job, use the Time Restrictions window for that job, as shown in Figure 12-2 on page 292. Note that the default is `No restrictions`, which means that there is no time restriction.

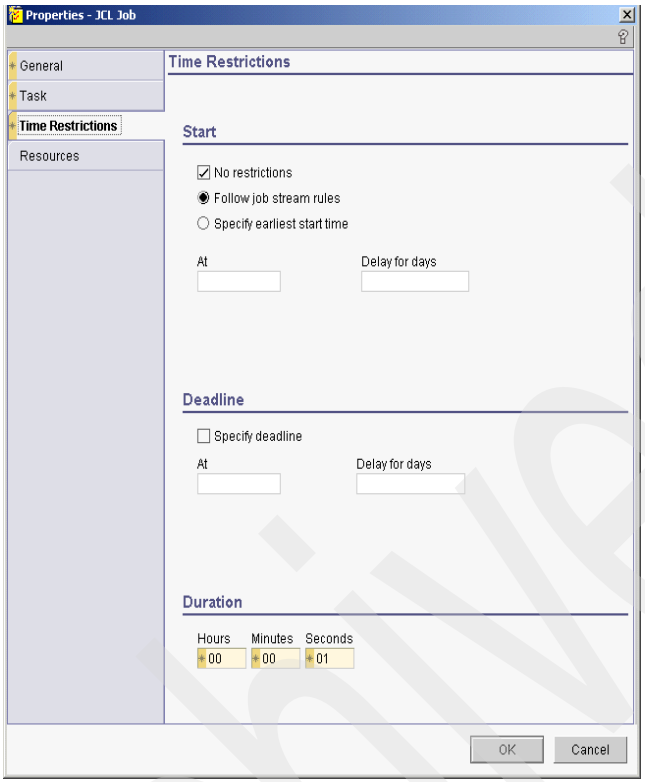


Figure 12-2 Time Restrictions window for a job

The main use of input arrival time is to resolve external dependencies. External dependencies are resolved backward in time using the input arrival time. It is also used for determining whether the job streams are included in the plan.

Input arrival time is part of a key for the job stream in the long-term and current plan. The key is date and time (hhmm), plus the job stream name. This makes it possible in Tivoli Workload Scheduler for z/OS to have multiple instances of the same job stream in the plan.

Note: This is not possible in the Tivoli Workload Scheduler Distributed product.

Input arrival time is also used when listing and sorting job streams in the long-term and current plan. It is called Start time in the Time Restrictions window of the Job Scheduling Console, as shown in Figure 12-3.

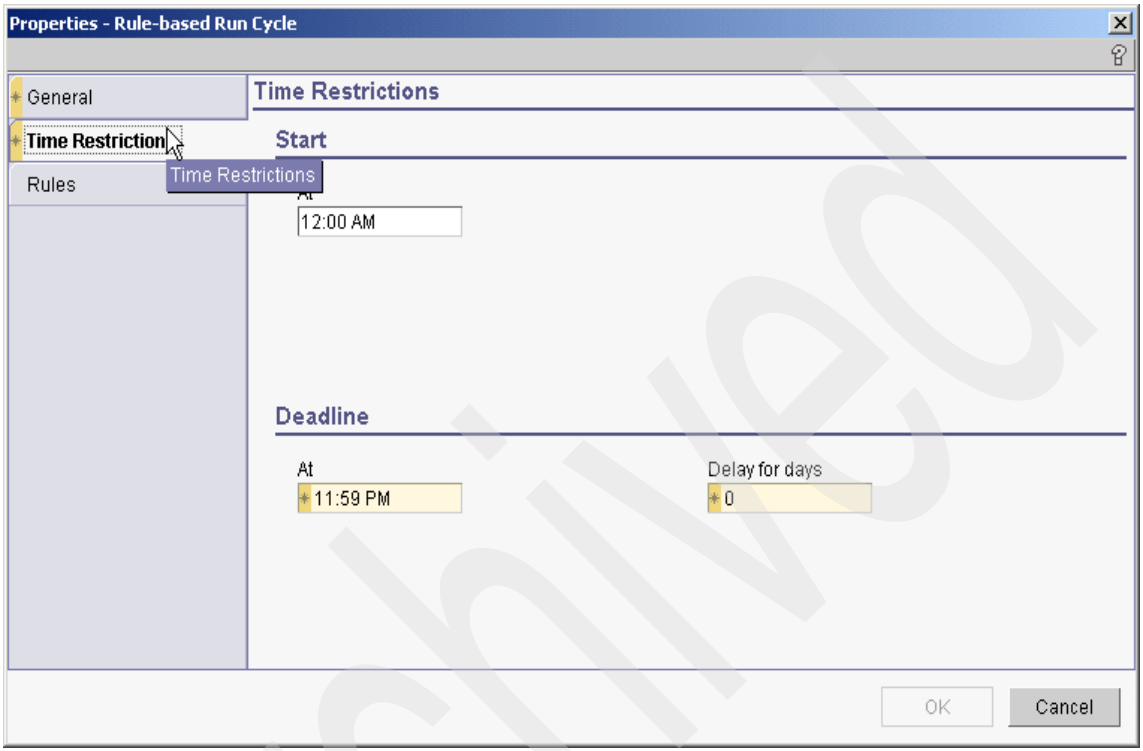


Figure 12-3 Start time (or input arrival time)

Note: Input arrival time (Start field in Figure 12-3) is not a required field in the JSC; the default is 12:00 AM (00:00), even if this field is blank.

We can explain this with an example.

Look at Figure 12-4. Assume that we have a 24-hour current plan that starts at 6:00 AM. Here, we have two job streams with the same name (JS11) in the plan. As required, these job streams, or occurrences, have different input arrival times: 9:00 AM and 5:00 PM, respectively. If there is no other dependency (time dependency or resource dependency), both job streams will run as soon as possible (when they have been selected as eligible by the WSA).

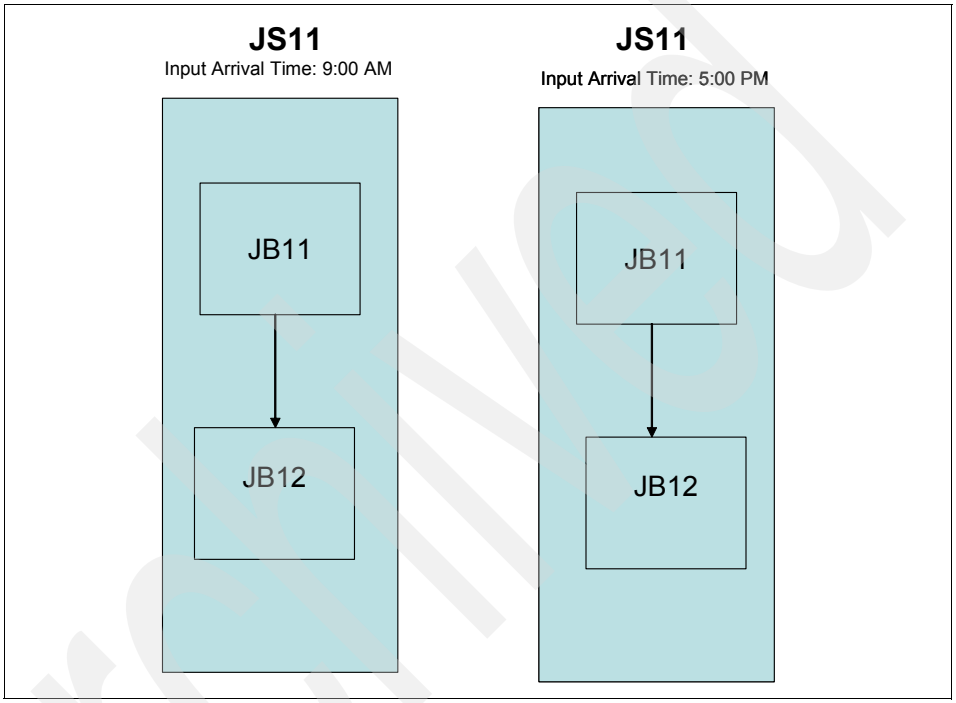


Figure 12-4 Two job streams with the same name

Now assume that there is another job stream (JS21) in the plan that has one job JB21. JB21 depends on successful completion of JB11 (Figure 12-5).

So far so good. But which JB11 will be considered as the predecessor of JB21? Here, the input arrival comes into play. To resolve this, Tivoli Workload Scheduler for z/OS will scan backward in time until the first predecessor occurrence is found. So, scanning backward from 3:00 PM (input arrival time of JB21), JB11 with the input arrival time of 9:00 AM will be found. (For readability, we show this job instance as JB11(1) and the other as JB11(2).)

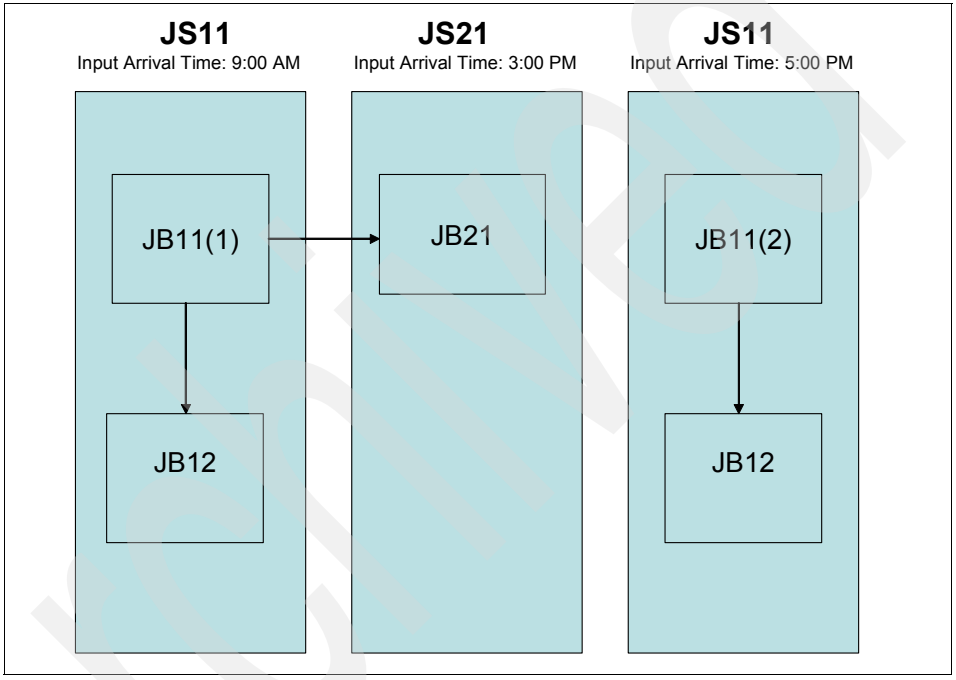


Figure 12-5 A new job steam JS21

With this logic, JB11(1) will be considered as the predecessor of JB21.

Tip: If the input arrival time of JB21 were, for example, 8:00 AM (or any time before 9:00 AM), Tivoli Workload Scheduler for z/OS would ignore the dependency of JB21 to JB11. When scanning backward from 8:00 AM, it would not be able to locate any occurrence of JB11.

Assume further that there is another job stream (JS01) in the plan that has one job, JB01 (Figure 12-6). This job stream has an input arrival time of 7:00 AM, and its job (JB01) has the following properties:

- It is a predecessor of JB11.
- It has a time dependency of 3:00 PM.

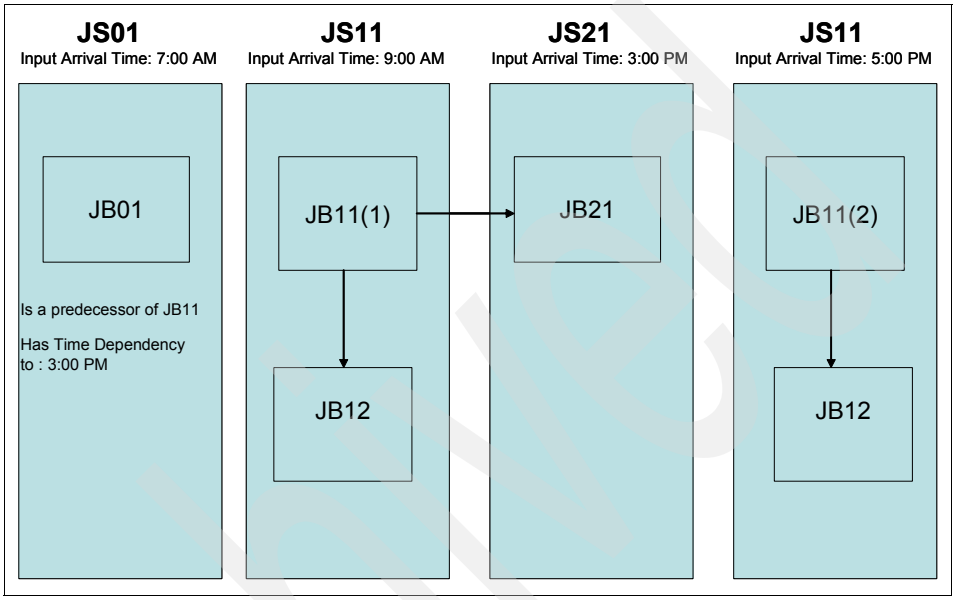


Figure 12-6 Job JS01 added to the plan

Assuming that current time is, for example, 8:00 AM, and there is no other dependency or no other factor that prevents its launch, which instance of JB11 will be eligible to run first: JB11(1) with an input arrival time 9:00 AM or JB11(2) with an input arrival time 5:00 PM?

The answer is JB11(2), although it has a later input arrival time. The reason is that Tivoli Workload Scheduler for z/OS will scan backward from 9:00 AM and calculate that JB01 is the predecessor of JB11(1). In that case, the dependency of JB11(2) will be ignored. This is an important concept. *External job dependencies in Tivoli Workload Scheduler for z/OS (and also on Tivoli Workload Scheduler Distributed) are ignored if these jobs are not in the current plan with the jobs that depend on them.* In other words, dependencies are not implied.

In most of the real-life implementations, the input arrival coding is not that complex, because usually only one instance of job stream (or occurrence) exists in the plan. In that case, there is no need for different input arrival time customizations. It could be same (or left default, which is 00:00) throughout all job streams. Nevertheless, the input arrival time is there for your use.

Note: Before finishing the input arrival discussion, we want to point out that if more than one run cycle defines a job stream with the same input arrival time, it constitutes a single job stream (or occurrence).

12.1.10 Exploit restart capabilities

When a job goes wrong, it can take some time to fix it, and perhaps the first fix attempt is just to rerun the job (as in the case of a 911 failure in a DB2 batch job).

Where this is the case, some simple statements in the job's JCL will cause it to try the initial recovery action. Therefore if it fails again, the operator can see that recovery was attempted and know immediately that this is a callout, rather than spending valuable time investigating the job's documentation.

12.2 Designing your batch network

In this section, we discuss how the way that you connect your batch jobs affects the processing time of the planning batch jobs. Some tests were done building plans having 100,000 or more jobs scheduled within them. The less connected the 100,000 jobs were to each other, the quicker the plan EXTEND job ran.

The networks built for the tests did not reflect the schedules that exist in real installations; they were built to examine how the external connectivity of the jobs affected the overall time needed to build the current plan. We ran the following tests:

- Test 1 consisted of a single job in the first application with 400 external dependencies to applications containing 250 operations.
- Test 2 consisted of a single job in the first application with 400 external applications that had 25 operations in each.
- Test 3 consisted of a single job in the first application with 10 external applications of one operation, each of which had 40 externals that had 250 operations in each.

The figures shown in Table 12-1 on page 298 come from the EQQDNTOP step from the current plan create job.

Table 12-1 Figures from the EQQDNTOP step from the current plan create job

	EXCP	CPU	SRB	CLOCK	SERV
Test 1	64546	16.34	.00	34.53	98057k
Test 2	32810	16.80	.00	24.73	98648k
Test 3	21694	15.70	.00	21.95	92566k

The less connected the 100,000 jobs were to each other, the lower the clock time and the lower the EXCP.

The probable reason for this is the alteration in the amount of the current plan that needs to be in storage for the whole network to be processed.

This can also be seen in the processing overheads associated with resolving an internal, as opposed to an external, dependency. When an operation completes in Tivoli Workload Scheduler for z/OS that has successors, both ends of the connection must be resolved.

In the case of an internal dependency, the dependant operation (job) will already be in storage with the rest of its application (job stream). The external dependency might be in an application that is not currently in storage and will have to be paged in to do the resolution (Figure 12-7).

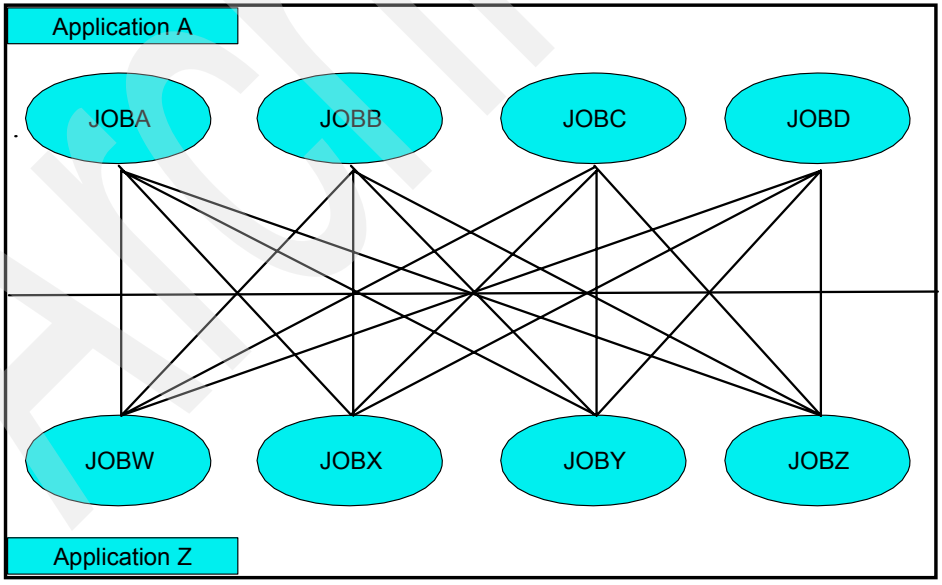


Figure 12-7 Sample applications (job streams) with internal and external dependencies

In terms of processing time, this does not equate to a large delay, but understanding this can help when making decisions about how to build your schedules.

Creating the minimal number of external dependencies is good practice anyway. Consider the flowcharts here: In the first (Figure 12-7 on page 298), we have 16 external dependencies. In the second (Figure 12-8), only one, just by adding a couple of operations on a dummy (non-reporting) workstation.

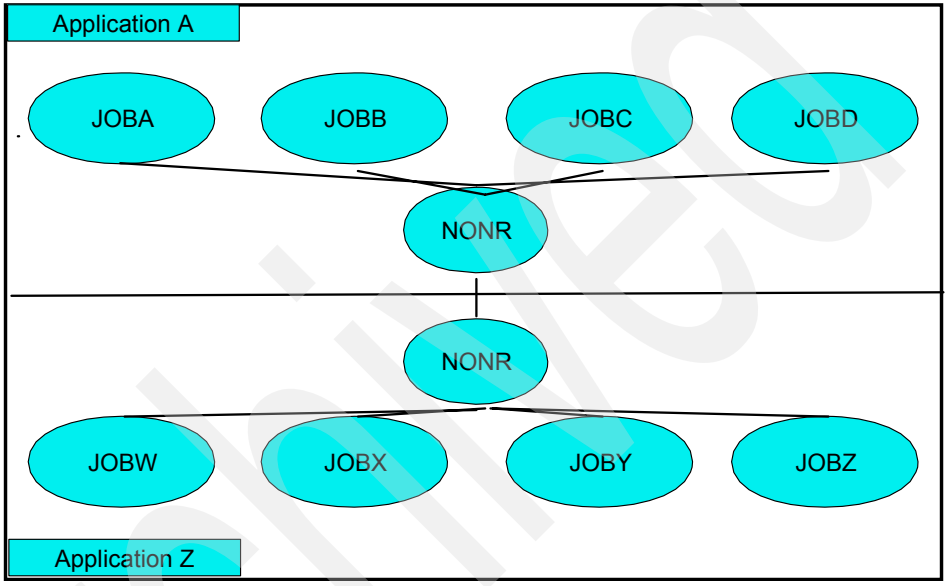


Figure 12-8 Adding a dummy (non-reporting) workstation

Good scheduling practices: Recommendations

The following recommendations are some best practices for creating job streams—in other words, good scheduling practices:

- Specify priority 9 only in exceptional circumstances and ensure that other priorities are used correctly.
- Ensure that operation durations are as accurate as possible.
- Set deadlines only in appropriate places.

These actions will ensure that the decisions made by Tivoli Workload Scheduler for z/OS when finding the next most urgent job are correct for your installation.

Inaccurate scheduling can cause many jobs to have the same internal priority to the scheduler. Preventing this will ensure that the most critical jobs are scheduled first, and will reduce unnecessary complexity in the schedules.

Build dependencies only where they really exist. Each operation that completes has to notify all of its successors. Keeping the dependencies direct will shorten this processing.

12.3 Moving JCL into the JS VSAM files

This section defines the best methods for improving the rate at which Tivoli Workload Scheduler for z/OS is able to move the JCL into the JCL VSAM repository (JS files).

The JCL is fetched from the JS VSAM files when a job is being submitted. If it is not found in the VSAM file, it is either fetched by EQQUX002 (user-written exit) or from the EQQJBLIB data set concatenation.

Normally, the JCL is moved into the VSAM files immediately prior to submission; however, this means that any delays in fetching the JCL are imbedded into the batch window. To avoid this delay, the JCL could be moved into the VSAM file earlier in the day, ready for submission. To be able to do this, you do need to know which jobs cannot be pre-staged (for example, any JCL that is built dynamically by some external process). You also need a PIF program that can do this selective pre-staging for you.

12.3.1 Pre-staging JCL tests: description

Tests were done to show how simple changes to the JCL library placement and library types, plus the use of other tools such as Tivoli Workload Scheduler for z/OS exits and LLA, can provide improvements in the JCL fetch time.

These tests used a program interface (PIF) REXX to fetch the JCL into the JS file. The tests were done fetching the JCL either by normal Tivoli Workload Scheduler for z/OS, through the EQQJBLIB concatenation, or by using EQQUX002. The JCL was held in either PDS or PDSE files. For some tests, the directories of the PDS libraries were held in LLA.

For each test, the current plan contained 100,000 jobs. The JCL was spread around four libraries with greater than 25,000 jobs in each. Nothing else was active in the CP, so the general service task, which handles PIF requests, got the best service possible.

12.3.2 Pre-staging JCL tests: results tables

Table 12-2 on page 301 shows the results of our first test with pre-staging the JCL. The results in this table are for tests done using slower DASD (9393-T82) with a smaller caching facility.

Table 12-2 Results of the pre-staging JCL tests (k = x1000)

	EXCP	CPU	SRB	CLOCK	SERV
4 x PDS	478k	26.49	0.08	375.45	68584k
4 x PDSE	455k	25.16	0.08	101.22	65174k
4 x PDS + EXITS	461k	25.55	0.08	142.62	65649k
4 x PDS + LLA	458k	25.06	0.08	110.92	64901k
4 x PDS + LLA + EXITS	455k	25.21	0.08	99.75	65355k
4 x PDSE + EXITS	455k	25.02	0.08	94.99	64871k

The results in Table 12-3 are for tests done using faster DASD with a very large cache. In fact, the cache was so large, we believe most if not all of the libraries were in storage for the PDSE and LLA tests.

Table 12-3 Results with faster DASD with a very large cache (k = x1000)

	EXCP	CPU	SRB	CLOCK	SERV
4 x PDS	457k	25.68	0.08	176.00	66497k
4 x PDSE	463k	25.20	0.08	129.81	65254k
4 x PDS + EXITS	455k	25.21	0.08	111.98	65358k
4 x PDS + LLA	455k	25.07	0.08	101.47	64915k
4 x PDS + LLA + EXITS	456k	24.98	0.08	95.12	64761k
4 x PDSE + EXITS	455k	25.02	0.08	94.99	64871k

One additional test was done using a facility provided by the EQQUX002 code we were using (Table 12-4). This enabled us to define some model JCL that was loaded into storage when the Tivoli Workload Scheduler for z/OS controller was started. When JCL was fetched, it was fetched from this storage version. The exit inserted the correct job name and other elements required by the model from data in the operations record (CPOP) in the current plan.

Table 12-4 Results using the EQQUX002 code shipped with this book (k = x1000)

	EXCP	CPU	SRB	CLOCK	SERV
EQQUX002	455k	25.37	0.08	95.49	65759k

As these results show, the quickest retrievals were possible when using PDSE files with the EQQUX000/002 exits, using PDS files with their directories in LLA with the EQQUX00/02 exits, or using the in-storage model JCL facility.

12.3.3 Pre-staging JCL conclusions

In the following sections, we discuss our pre-staging JCL conclusions.

Using PDSE files for JCL

From the previously described information, the obvious course of action would be to move all the JCL files into PDSE files. However, if your JCL is small, this wastes space. In a PDSE, a single member takes up one 4096-byte page at a minimum. A page cannot be shared by more than a single member. This makes this a relatively costly option when considering disk usage. We also found that accessing a PDSE library through our TSO sessions for edit or browse took considerably longer than accessing the PDS files.

For example, our JCL consisted of four records: A jobcard that continued over two lines, an exec card, and a steplib. For approximately 25,000 members, that equated to 3330 tracks for a PDSE and only 750 tracks for a PDS.

Obviously from a space perspective, the cheapest option was the model JCL, because all of our jobs followed the same model, so only one four-line member was needed.

The use of LLA for the PDS directories

Some of the fetch time is attributable to the directory search. By placing the JCL libraries under LLA, the directory search times are greatly improved.

The issue with doing this is the maintenance of the JCL libraries. Because these libraries are allocated to a long-running task, it is advisable to stop the controller prior to doing an LLA refresh. (Alternatively, you can use the LLA UPDATE command, as shown in the Note on page 303.)

One alternative is to have a very small override library, that is not in LLA, placed in the top of the EQQJBLIB concatenation for changes to JCL. This way, the need for a refresh at every JCL change is avoided. The LLA refresh could then be scheduled once a day or once a week, depending on the rate of JCL change.

Note: An LLA refresh will allow changes made to the source libraries to be picked up, but you can also use an LLA update command:

```
F LLA,UPDATE=xx
```

Here the CSVLLAxx member contains a NOFREEZE statement for the PDS that needs to be updated.

Then another LLA update command, **F LLA,UPDATE=yy**, can be issued, where CSVLLAyy contains a FREEZE statement for the source PDS library. By doing the NOFREEZE and then the FREEZE commands, the need for an LLA refresh is avoided.

The use of EQQUX000 and EQQUX002

The use of the exits saves the directory search from potentially running down all the libraries in the EQQJCLIB concatenation by directing the fetch to a specific library. In our tests, these libraries were all quite large, but had we used many more libraries with fewer members in each, we would have recovered more time.

The use of model JCL is also one of the fastest methods of populating the JS file. This has an additional benefit. If you can, as we did, have a single model of JCL that is valid for many jobs, you also remove all those jobs from the EQQJBLIB or exit DD concatenations.

12.4 Recommendations

To ensure that Tivoli Workload Scheduler for z/OS performs well, both in terms of dialog response times and job submission rates, the following recommendations should be implemented. However, it should be noted that although these enhancements can improve the overall throughput of the base product, the amount of work that Tivoli Workload Scheduler for z/OS has to process in any given time frame will always be the overriding factor. The recommendations are listed in the sequence that provides the most immediate benefit.

12.4.1 Pre-stage JCL

Using a program interface staging program, move as much JCL as possible to the JS file before it reaches a ready state. The preferred method is to use a Tivoli Workload Scheduler for z/OS PIF program.

After the JCL has been staged, the BACKUP JS command should be used to clean up the CI and CA splits caused by so many inserts. This also improves the general performance of the JS VSAM files.

It is not necessary to back up this file very often; two to four times a day is sufficient, especially if the majority of the activity takes place once a day during the pre-staging process.

12.4.2 Optimize JCL fetch: LLA

Place the job libraries, defined by the data definition (DD) statement EQQJBLIB in the Tivoli Workload Scheduler for z/OS started task, in LLA or a PDS management product. For LLA, this is achieved by using the LIBRARY and FREEZE options.

Updates to libraries in LLA will not be accessible to the controller until an LLA REFRESH or LLA UPDATE command (see the Note on page 303) has been issued for the library in question. A simple technique to cover this is to have a small UPDATE library concatenated ahead of the production job library that would not be placed in LLA. On a regular basis (for example, weekly), move all updated JCL from the update to the production library at the same time as a refresh LLA.

Note: The use of LLA for PDS libraries is dependent on the level of z/OS UNIX System Services installed.

12.4.3 Optimize JCL fetch: exits

Implement exit EQQUX002 to reduce JCL location time by reading the JCL from a specific data definition statement based on a value or values available to the exit. Examples include application name, job name, jobclass, or forms type.

Note: Libraries defined specifically for use by the EQQUX002 exit should also be placed within LLA or an equivalent if possible.

In addition, implement exit EQQUX000 to improve EQQUX002 performance by moving all open/close routines to Tivoli Workload Scheduler for z/OS startup, instead of doing it each time EQQUX002 is called. The moving of JCL libraries under LLA and the introduction of EQQUX002 and EQQUX000 provides significant performance improvements. The use of LLA provides the greatest single improvement; however, this is not always practical, especially where an installation's JCL changes frequently. The exits alone can cause a significant improvement and using them with LLA is the most beneficial.

Ensure that all the Tivoli Workload Scheduler for z/OS libraries (including VSAM) are placed on the fastest possible volumes.

12.4.4 Best practices for tuning and use of resources

The following list describes the best practices for the tuning and use of system and Tivoli Workload Scheduler for z/OS resources for optimizing Tivoli Workload Scheduler for z/OS throughput:

- Ensure that the JS file does not go into extents and that CI and CA splits are kept to a minimum. This will ensure that the JCL repository does not become fragmented, which leads to delays in job submission.
- Ensure that the JS file is backed up periodically, at times that are useful to your installation (see 12.3.1, “Pre-staging JCL tests: description” on page 300).
- Enter a value of NO in the MAXJSFILE initialization parameter to avoid Tivoli Workload Scheduler for z/OS initiating the JS backups. This also places a lock on the current plan and is often the longest single activity undertaken by the NMM. Run a batch job or TSO command regularly to execute the BACKUP (JS) command instead.
- Clear out the JS file at regular intervals. It has a tendency to grow, because jobs that are run only once are never removed. A sample in SEQQSAMP (EQQPIFJX) can be used to delete items that are older than required.
- Consider all methods of reducing the number of members, and their size, within production JCL libraries.
- Regularly clean the libraries and remove all redundant members.
- Whenever possible, call procedures rather than maintain large JCL streams in Tivoli Workload Scheduler for z/OS libraries. Use JCL variables to pass specific details to the procedures, where procedural differences are based on data known to Tivoli Workload Scheduler for z/OS, such as workstation.
- Allow the Tivoli Workload Scheduler for z/OS exit EQQUX002 to create RDR JCL from a model. This idea is useful when, for example, several of the members in the job library (especially if you have hundreds or thousands) execute a procedure name that is the same as the job name (or can be derived from it). Replacing the several members with just a few model members (held in storage) and having the exit modify the EXEC card would reduce the size of the job library and therefore the workstation analyzer overhead during JCL fetch times.

12.4.5 Implement EQQUX004

Implement EQQUX004 to reduce the number of events that the event manager has to process.

Running non-production (or non-Tivoli Workload Scheduler for z/OS controlled) jobs on a processor that has a tracker started will generate events of no

consequence that have to be written to the event data set and passed on to the controller. These will have to be checked by the controller's event manager against the current plan and discarded.

Removing these events can improve the overall performance of the controller by lessening its overhead.

12.4.6 Review your tracker and workstation setup

Where possible, workstations should direct their work to trackers for submission, especially where more than one system image is being controlled. This saves the controller the overhead of passing all the jobs to a single internal reader, which might itself prove to be a submission bottleneck. Delays would also be introduced in using some other router on the system (NJE) to pass the job to the appropriate execution system.

Consideration should be given to the method of communication used between the controller and the trackers. Of the three methods, XCF gives the best performance; however, its use is possible only in installations with the right hardware and software configurations. Using VTAM (the NCF task) is second in the performance stakes, with shared DASD being the slowest because it is I/O intensive.

12.4.7 Review initialization parameters

Review your Tivoli Workload Scheduler for z/OS parameters and ensure that no unnecessary overhead has been caused by parameters that are not required by your installation, such as:

- Set PRINTEVENTS(NO) if printing is not tracked.
- Do not use STATMSG except when needing to analyze your system or when collecting historical data.

12.4.8 Review your z/OS UNIX System Services and JES tuning

Ensure that your system is tuned to cope with the numbers of jobs being scheduled by Tivoli Workload Scheduler z/OS. It does no good to be able to schedule 20 jobs a second if the JES parameters are throttling back the systems and only allowing five jobs per second onto the JES queues.

Specifically review *System/390 MVS Parallel Sysplex Continuous Availability Presentation Guide*, SG24-4502, paying special attention to the values coded for HOLD and DORMANCY.

Part 2

Tivoli Workload Scheduler for z/OS end-to-end scheduling

In this part we introduce IBM Tivoli Workload Scheduler for z/OS end-to-end scheduling.

13

Introduction to end-to-end scheduling

In this chapter we describe end-to-end scheduling in Tivoli Workload Scheduler for z/OS and provide some background architecture about the end-to-end environment. We also cover the positioning of end-to-end scheduling by comparing the pros and cons of alternatives for customers for mainframe and distributed scheduling needs.

This chapter has the following sections:

- Introduction to end-to-end scheduling
- The terminology used in this book
- Tivoli Workload Scheduler architecture
- End-to-end scheduling: how it works
- Comparing enterprise-wide scheduling deployment scenarios

13.1 Introduction to end-to-end scheduling

End-to-end scheduling means scheduling workload across all computing resources in your enterprise, from the mainframe in your data center, to the servers in your regional headquarters, all the way to the workstations in your local office. The Tivoli Workload Scheduler for z/OS end-to-end scheduling solution is a system whereby scheduling throughout the network is defined, managed, controlled, and tracked from a single IBM mainframe or sysplex.

End-to-end scheduling requires using two different programs: Tivoli Workload Scheduler for z/OS on the mainframe, and Tivoli Workload Scheduler (or Tivoli Workload Scheduler Distributed) on other operating systems (UNIX, Windows®, and OS/400®). This is shown in Figure 13-1.

Note: Throughout this book, we refer to the Tivoli Workload Scheduler Distributed product as Tivoli Workload Scheduler.

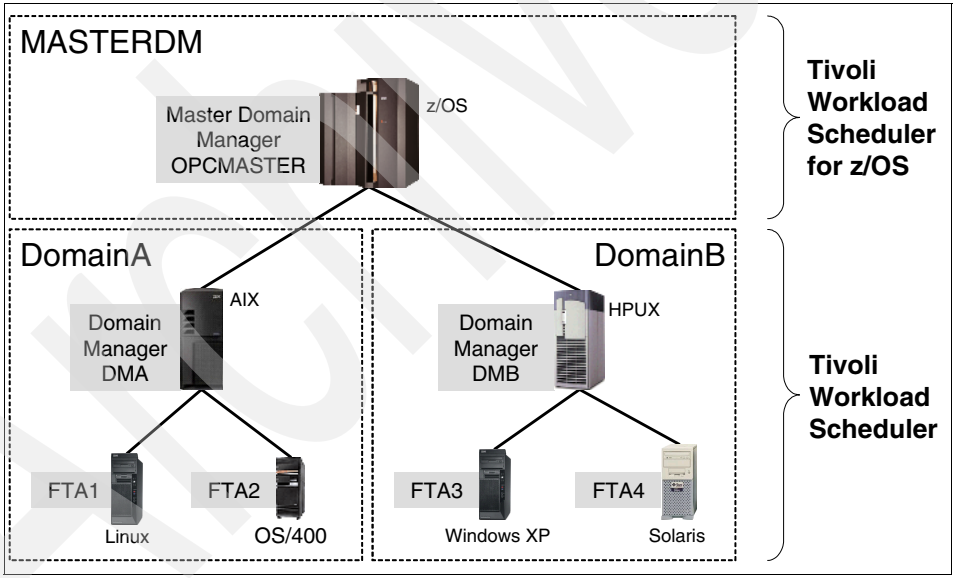


Figure 13-1 Both schedulers are required for end-to-end scheduling

Despite the similar names, Tivoli Workload Scheduler for z/OS and Tivoli Workload Scheduler are quite different and have distinct histories. Tivoli Workload Scheduler for z/OS was originally called OPC (Operations Planning & Control). It was developed by IBM in the early days of the mainframe. Actually, OPC still exists in Tivoli Workload Scheduler for z/OS.

Tivoli Workload Scheduler for z/OS and Tivoli Workload Scheduler (sometimes called Tivoli Workload Scheduler for Distributed) have slightly different ways of working, and the programs have many features in common. IBM has continued development of both programs toward the goal of providing closer and closer integration between them. The reason for this integration is simple: to facilitate an integrated scheduling system across all operating systems.

It should be obvious that end-to-end scheduling depends on using the mainframe as the central point of control for the scheduling network. There are other ways to integrate scheduling between z/OS and other operating systems.

Tivoli Workload Scheduler is descended from the Unison Maestro™ program. Unison Maestro was developed by Unison Software on the Hewlett-Packard MPE operating system. It was then ported to UNIX and Windows. In its various manifestations, Tivoli Workload Scheduler has a 19-year track record. During the processing day, Tivoli Workload Scheduler manages the production environment and automates most operator activities. It prepares jobs for execution, resolves interdependencies, and launches and tracks each job. Because jobs begin as soon as their dependencies are satisfied, idle time is minimized. Jobs never run out of sequence. If a job fails, Tivoli Workload Scheduler can handle the recovery process with little or no operator intervention.

13.1.1 Overview of Tivoli Workload Scheduler

As with Tivoli Workload Scheduler for z/OS, there are two basic aspects to job scheduling in Tivoli Workload Scheduler: The *database* and the *plan*. The database contains all definitions for scheduling objects, such as jobs, job streams, resources, days and times jobs should run, dependencies, and workstations. It also holds statistics of job and job stream execution, as well as information on the user ID that created an object and when an object was last modified. The plan contains all job scheduling activity planned for a period of one day. In Tivoli Workload Scheduler, the plan is created every 24 hours and consists of all the jobs, job streams, and dependency objects that are scheduled to execute for that day. Job streams that do not complete successfully can be carried forward into the next day's plan.

13.1.2 Tivoli Workload Scheduler network

A typical Tivoli Workload Scheduler network consists of a *master domain manager*, *domain managers*, and *fault-tolerant agents*. The master domain manager, sometimes referred to as just the *master*, contains the centralized database files that store all defined scheduling objects. The master creates the plan, called *Symphony*, at the start of each day.

Each domain manager is responsible for distribution of the plan to the fault-tolerant agents (FTAs) in its domain. A domain manager also handles resolution of dependencies between FTAs in its domain.

Fault-tolerant agents, the workhorses of a Tivoli Workload Scheduler network, are where most jobs are run. As their name implies, fault-tolerant agents are *fault tolerant*. This means that in the event of a loss of communication with the domain manager, FTAs are capable of resolving local dependencies and launching their jobs without interruption. FTAs are capable of this because each FTA has its own copy of the Symphony plan. The Symphony plan contains a complete set of scheduling instructions for the production day. Similarly, a domain manager can resolve dependencies between FTAs in its domain even in the event of a loss of communication with the master, because the domain manager's plan receives updates from all subordinate FTAs and contains the authoritative status of all jobs in that domain.

The master domain manager is updated with the status of all jobs in the entire IBM Tivoli Workload Scheduler network. Logging and monitoring of the IBM Tivoli Workload Scheduler network is performed on the master.

Starting with Tivoli Workload Scheduler V7.0, a new Java™-based graphical user interface was made available to provide an easy-to-use interface to Tivoli Workload Scheduler. This new GUI is called the Job Scheduling Console (JSC). The current version of JSC has been updated with several functions specific to Tivoli Workload Scheduler. The JSC provides a common interface to both Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS.

13.2 The terminology used in this book

The Tivoli Workload Scheduler V8.2 suite comprises two somewhat different software programs, each with its own history and terminology. For this reason, there are sometimes two different and interchangeable names for the same thing. Other times, a term used in one context can have a different meaning in another context. To help clear up this confusion, we now introduce some of the terms and acronyms that will be used throughout the book. In order to make the terminology used in this book internally consistent, we adopted a system of terminology that may be a bit different than that used in the product documentation. So take a moment to read through this list, even if you are already familiar with the products.

IBM Tivoli Workload Scheduler V8.2 suite

The suite of programs that includes Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS. These programs are used together to make end-to-end

scheduling work. Sometimes called just Tivoli Workload Scheduler.

IBM Tivoli Workload Scheduler
The version of Tivoli Workload Scheduler that runs on UNIX, OS/400, and Windows operating systems, as distinguished from Tivoli Workload Scheduler for z/OS, a somewhat different program. Sometimes called IBM Tivoli Workload Scheduler Distributed. Tivoli Workload Scheduler is based on the old Maestro program.

IBM Tivoli Workload Scheduler for z/OS
The version of Tivoli Workload Scheduler that runs on z/OS, as distinguished from Tivoli Workload Scheduler (by itself, without the *for z/OS* specification). Tivoli Workload Scheduler for z/OS is based on the old OPC (Operations Planning & Control) program.

Master The top level of the Tivoli Workload Scheduler or Tivoli Workload Scheduler for z/OS scheduling network. Also called the master domain manager, because it is the domain manager of the MASTERDM (top-level) domain.

Domain manager The agent responsible for handling dependency resolution for subordinate agents. Essentially an FTA with a few extra responsibilities.

Backup domain manager
A fault-tolerant agent or domain manager capable of assuming the responsibilities of its domain manager for automatic workload recovery.

Fault-tolerant agent (FTA) An agent that keeps its own local copy of the plan file and can continue operation even if the connection to the parent domain manager is lost. In Tivoli Workload Scheduler for z/OS, FTAs are referred to as *fault-tolerant workstations*.

Standard agent A workstation that launches jobs only under the direction of its domain manager.Tivoli Workload Scheduler

Extended agent A logical workstation that enables you to launch and control jobs on other systems and applications, such as PeopleSoft, Oracle Applications, SAP, and MVS JES2 and JES3.

Scheduling engine A Tivoli Workload Scheduler engine or Tivoli Workload Scheduler for z/OS engine.

IBM Tivoli Workload Scheduler engine
The part of Tivoli Workload Scheduler that does actual

scheduling work, as distinguished from the other components that are related primarily to the user interface (for example, the Tivoli Workload Scheduler Connector). Essentially the part of Tivoli Workload Scheduler that is descended from the old Maestro program.

IBM Tivoli Workload Scheduler for z/OS engine
The part of Tivoli Workload Scheduler for z/OS that does actual scheduling work, as distinguished from the other components that are related primarily to the user interface (for example, the Tivoli Workload Scheduler for z/OS Connector). Essentially the controller plus the server.

IBM Tivoli Workload Scheduler for z/OS controller
This is the component that runs on the controlling system, and that contains the tasks that manage the plans and the databases.

IBM Tivoli Workload Scheduler for z/OS tracker
The tracker acts as a communication link between the system it runs on and the controller.

LTP (long-term plan) A high-level plan for system activity that covers a period of at least one day, and not more than four years.

CP (current plan) A detailed plan or schedule of system activity that covers at least one minute, and not more than 21 days. Typically a current plan will cover one or two days.

WS (workstation) A unit, place or group that performs specific data processing functions. A logical place where work occurs in an operations department. Tivoli Workload Scheduler for z/OS requires that you define the following characteristic for each workstation: the type of work it does (computer, printer, or general), the quantity of work it can handle at any particular time, and the times it is active. The activity that occurs at each workstation is called an operation.

Dependency An operation that is dependent on either an internal or external operation; it must complete successfully before the operation can start.

IBM Tivoli Workload Scheduler for z/OS server
The part of Tivoli Workload Scheduler for z/OS that is based on the UNIX IBM Tivoli Workload Scheduler code. Runs in UNIX System Services (USS) on the mainframe.

JSC Job Scheduling Console, the common graphical user interface (GUI) to both the IBM Tivoli Workload Scheduler

	and IBM Tivoli Workload Scheduler for z/OS scheduling engines.
Connector	A small program that provides an interface between the common GUI (Job Scheduling Console) and one or more scheduling engines. The connector translates to and from the different “languages” used by the different scheduling engines.
JSS	Job Scheduling Services. Essentially a library that is used by the connectors.
TMF	Tivoli Management Framework. Also called just the Framework.

13.3 Tivoli Workload Scheduler architecture

Tivoli Workload Scheduler helps you plan every phase of production. During the processing day, its production control programs manage the production environment and automate most operator activities. Tivoli Workload Scheduler prepares jobs for execution, resolves interdependencies, and launches and tracks each job. Because jobs start running as soon as their dependencies are satisfied, idle time is minimized and throughput is improved. Jobs never run out of sequence in Tivoli Workload Scheduler. If a job ends in error, Tivoli Workload Scheduler handles the recovery process with little or no operator intervention.

Tivoli Workload Scheduler is composed of three major parts:

- Tivoli Workload Scheduler engine

  The Tivoli Workload Scheduler engine is installed on every non-mainframe workstation in the scheduling network (UNIX, Windows, and OS/400 computers). When the engine is installed on a workstation, it can be configured to play a specific role in the scheduling network. For example, the engine can be configured to be a master domain manager, a domain manager, or a fault-tolerant agent. In an ordinary Tivoli Workload Scheduler network, there is a single master domain manager at the top of the network. However, in an end-to-end scheduling network, *there is no master domain manager*. Instead, its functions are instead performed by the Tivoli Workload Scheduler for z/OS engine, installed on a mainframe.

- Tivoli Workload Scheduler Connector

  The connector “connects” the Job Scheduling Console to Tivoli Workload Scheduler, routing commands from JSC to the Tivoli Workload Scheduler engine. In an ordinary Tivoli Workload Scheduler network, the Tivoli Workload Scheduler Connector is usually installed on the master domain manager. In an end-to-end scheduling network, there is no master domain manager so the

Connector is usually installed on the first-level domain managers. The Tivoli Workload Scheduler Connector can also be installed on other domain managers or fault-tolerant agents in the network.

The connector software is installed on top of the Tivoli Management Framework, which must be configured as a Tivoli Management Region server or managed node. The connector software cannot be installed on a Tivoli Management Regions (TMR) endpoint.

- Job Scheduling Console (JSC)

  JSC is the Java-based graphical user interface for the Tivoli Workload Scheduler suite. The Job Scheduling Console runs on any machine from which you want to manage Tivoli Workload Scheduler plan and database objects. Through the Tivoli Workload Scheduler Connector, it provides the functions of the command-line programs conman and composer. The Job Scheduling Console can be installed on a desktop workstation or laptop, as long as the JSC has a TCP/IP link with the machine running the Tivoli Workload Scheduler Connector. Using the JSC, operators can schedule and administer Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS over the network. More on the JSC, including installation, can be found in Chapter 16, “Using the Job Scheduling Console with Tivoli Workload Scheduler for z/OS” on page 481.

13.3.1 The Tivoli Workload Scheduler network

A Tivoli Workload Scheduler network is made up of the workstations, or CPUs, on which jobs and job streams are run.

A Tivoli Workload Scheduler network contains at least one Tivoli Workload Scheduler domain, the master domain, in which the master domain manager is the management hub. It is the master domain manager that manages the databases, and it is from the master domain manager that you define new objects in the databases. Additional domains can be used to divide a widely distributed network into smaller, locally managed groups.

In the simplest configuration, the master domain manager maintains direct communication with all of the workstations (fault-tolerant agents) in the Tivoli Workload Scheduler network. All workstations are in the same domain, MASTERDM (Figure 13-2).

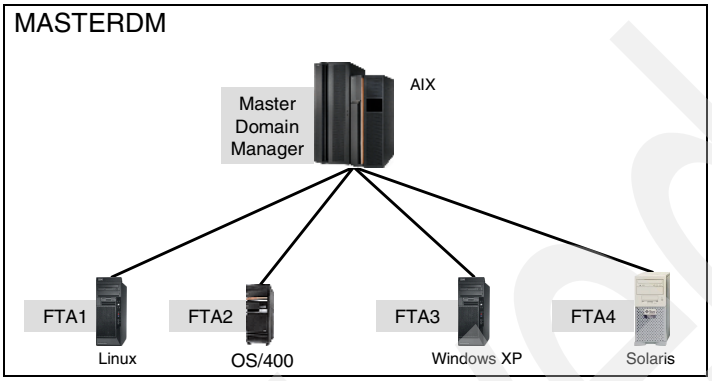


Figure 13-2 A sample Tivoli Workload Scheduler network with only one domain

Using multiple domains reduces the amount of network traffic by reducing the communications between the master domain manager and the other computers in the network. Figure 13-3 on page 318 depicts an example of a Tivoli Workload Scheduler network with three domains. In this example, the master domain manager is shown as an AIX system, but it does not have to be on an AIX system; it can be installed on any of several different platforms, including AIX, Linux®, Solaris™, HPUX, and Windows. Figure 13-3 on page 318 is only an example that is meant to give an idea of a typical Tivoli Workload Scheduler network.

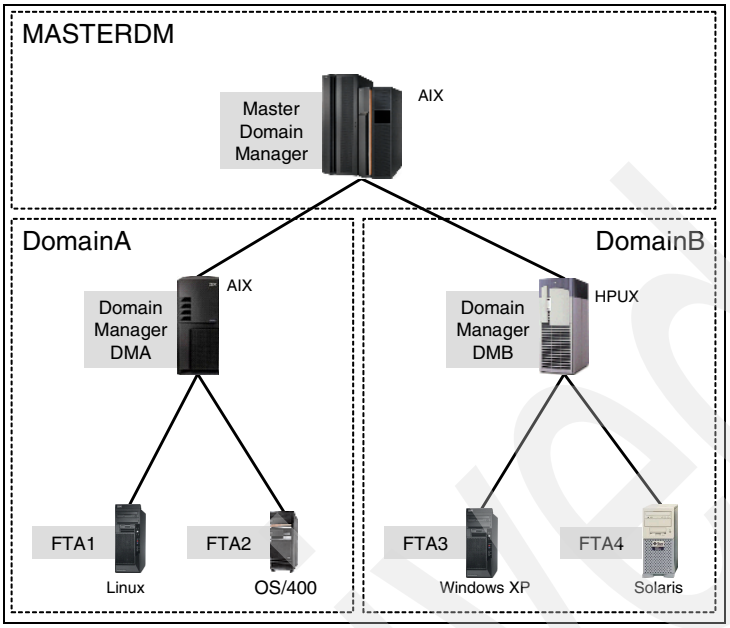


Figure 13-3 Tivoli Workload Scheduler network with three domains

In this configuration, the master domain manager communicates directly only with the subordinate domain managers. The subordinate domain managers communicate with the workstations in their domains. In this way, the number of connections from the master domain manager are reduced. Multiple domains also provide fault tolerance: If the link from the master is lost, a domain manager can still manage the workstations in its domain and resolve dependencies between them. This limits the impact of a network outage. Each domain may also have one or more backup domain managers that can take over if the domain manager fails.

Before the start of each day, the master domain manager creates a plan for the next 24 hours. This plan is placed in a production control file, named Symphony. Tivoli Workload Scheduler is then restarted throughout the network, and the master domain manager sends a copy of the Symphony file to each of the subordinate domain managers. Each domain manager then sends a copy of the Symphony file to the fault-tolerant agents in that domain.

After the network has been started, scheduling events such as job starts and completions are passed up from each workstation to its domain manager. The domain manager updates its Symphony file with the events and then passes the events up the network hierarchy to the master domain manager. The events are then applied to the Symphony file on the master domain manager. Events from all workstations in the network will be passed up to the master domain manager.

In this way, the master's Symphony file contains the authoritative record of what has happened during the production day. The master also broadcasts the changes down throughout the network, updating the Symphony files of domain managers and fault-tolerant agents that are running in full status mode.

It is important to remember that Tivoli Workload Scheduler does not limit the number of domains or levels (the hierarchy) in the network. There can be as many levels of domains as is appropriate for a given computing environment. The number of domains or levels in the network should be based on the topology of the physical network where Tivoli Workload Scheduler is installed. Most often, geographical boundaries are used to determine divisions between domains.

Figure 13-4 shows an example of a four-tier Tivoli Workload Scheduler network:

1. Master domain manager, MASTERDM
2. DomainA and DomainB
3. DomainC, DomainD, DomainE, FTA1, FTA2, and FTA3
4. FTA4, FTA5, FTA6, FTA7, FTA8, and FTA9

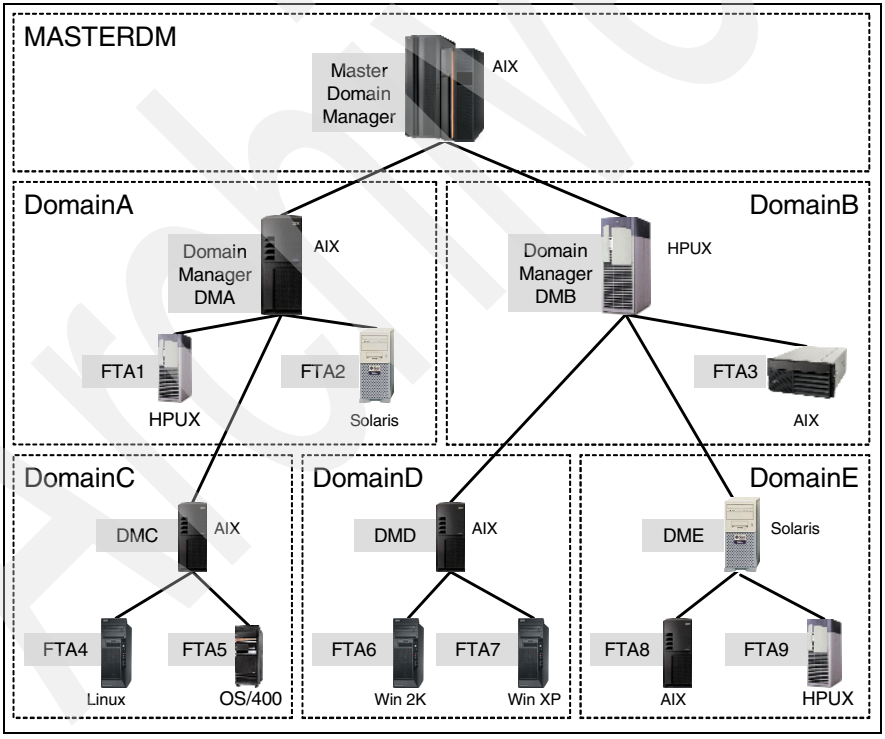


Figure 13-4 A multi-tiered Tivoli Workload Scheduler network

13.3.2 Tivoli Workload Scheduler workstation types

For most cases, workstation definitions refer to physical workstations. However, in the case of extended and network agents, the workstations are logical definitions that must be hosted by a physical Tivoli Workload Scheduler workstation.

There are several different types of Tivoli Workload Scheduler workstations:

- Master domain manager (MDM)

  The domain manager of the topmost domain of a Tivoli Workload Scheduler network. It contains the centralized database of all defined scheduling objects, including all jobs and their dependencies. It creates the plan at the start of each day, and performs all logging and reporting for the network. The master distributes the plan to all subordinate domain managers and fault-tolerant agents. In an end-to-end scheduling network, the Tivoli Workload Scheduler for z/OS engine (controller) acts as the master domain manager. In Figure 13-5, the master domain manager is shown as an AIX system, but could be any of several different platforms such as Linux, Solaris, HPUX and Windows, to name a few.

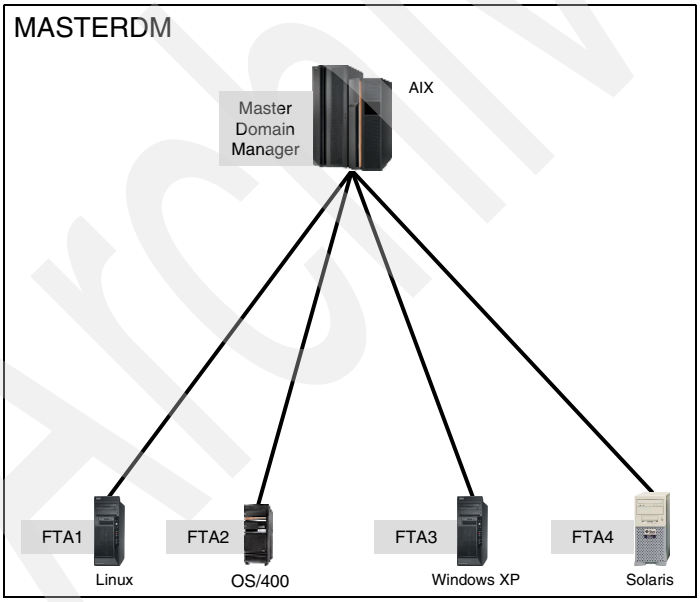


Figure 13-5 IBM Tivoli Workload Scheduler with three domains

- Domain manager (DM)

  The management hub in a domain. All communications to and from the agents in a domain are routed through the domain manager, which can

resolve dependencies between jobs in its subordinate agents. The copy of the plan on the domain manager is updated with reporting and logging from the subordinate agents. In Figure 13-6 the master domain manager communicates directly only with the subordinate domain managers. The domain managers then communicate with the workstations in their domains.

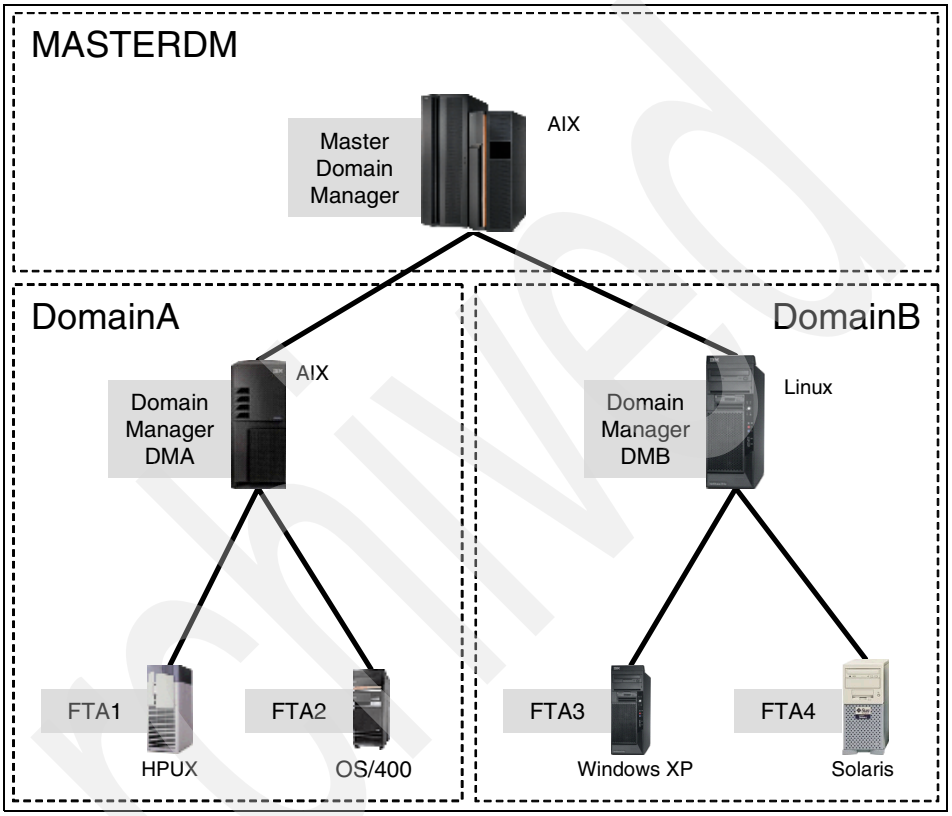


Figure 13-6 Master domain and domain managers

- Backup domain manager

  A fault-tolerant agent that is capable of assuming the responsibilities of its domain manager. The copy of the plan on the backup domain manager is updated with the same reporting and logging information as the domain manager plan.

- Fault-tolerant agent (FTA)

  A workstation that is capable of resolving local dependencies and launching its jobs in the absence of a domain manager. It has a local copy of the plan generated in the master domain manager. It is also called a fault-tolerant workstation.

- Standard agent (SA)

  A workstation that launches jobs only under the direction of its domain manager.

- Extended agent (XA)

  A logical workstation definition that enables one to launch and control jobs on other systems and applications. Tivoli Workload Scheduler for Applications includes extended agent methods for the following systems: SAP R/3, Oracle Applications, PeopleSoft, CA7, JES2, and JES3.

It is important to remember that domain manager FTAs, including the master domain manager FTA and backup domain manager FTAs, are FTAs with some extra responsibilities. The servers with these FTAs can, and most often will, be servers where you run normal batch jobs that are scheduled and tracked by Tivoli Workload Scheduler. This means that these servers do not have to be servers dedicated only for Tivoli Workload Scheduler work. The servers can still do some other work and run some other applications.

Tip: You should not choose to use one of your busiest servers as one of your Tivoli Workload Scheduler domain managers of first level.

More on Tivoli Workload Scheduler extended agents

Tivoli Workload Scheduler extended agents (XA) are used to extend the job scheduling functions of Tivoli Workload Scheduler to other systems and applications. An extended agent is defined as a workstation that enables you to launch and control jobs on other systems and applications. An extended agent is defined as a workstation that has a host and an access method.

Note: Tivoli Workload Scheduler extended agents are packaged as a licensed product called IBM Tivoli Workload Scheduler for Applications. IBM Tivoli Workload Scheduler is a prerequisite of this product.

The host is another IBM Tivoli Workload Scheduler workstation such as a fault-tolerant agent (FTA) or a standard agent (SA) that resolves dependencies and issues job launch requests via the method.

The access method is an IBM-supplied, user-supplied or program that is executed by the hosting workstation whenever Tivoli Workload Scheduler, either through its command line or the Tivoli Job Scheduling Console, needs to interact with the external system. IBM Tivoli Workload Scheduler for Applications includes the following access methods: Oracle Applications, SAP R/3, PeopleSoft, CA7, Tivoli Workload Scheduler z/OS, JES2 and JES3.

To launch and monitor a job on an extended agent, the host executes the access method, passing it job details as command line options. The access method communicates with the external system to launch the job and returns the status of the job.

An extended agent workstation is only a logical entity related to an access method hosted by the physical Tivoli Workload Scheduler workstation. More than one extended agent workstation can be hosted by the same Tivoli Workload Scheduler workstation and rely on the same access method. The x-agent is defined in a standard Tivoli Workload Scheduler workstation definition, which gives the x-agent a name and identifies the access method.

To launch a job in an external environment, Tivoli Workload Scheduler executes the extended agent access method, which provides the extended agent workstation name and information about the job. The method looks at the corresponding file named <WORKSTATION_NAME>_<method_name>.opts to determine which external environment instance to connect to. The access method can then launch jobs on that instance and monitor them through completion writing job progress and status information in the standard list file of the job.

Figure 13-7 shows the connection between TWS and an extended agent.

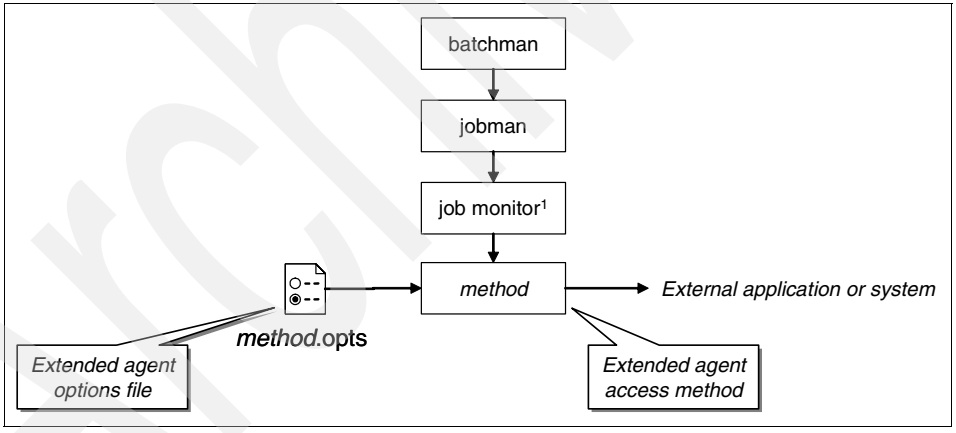


Figure 13-7 Extended agent processing

Find more about extended agent processing in *Implementing IBM Tivoli Workload Scheduler V 8.2 Extended Agent for IBM Tivoli Storage Manager*, SG24-6696.

Note: Extended agents can be used to run jobs also in an end-to-end environment, where their scheduling and monitoring is performed from a Tivoli Workload Scheduler for z/OS controller.

13.4 End-to-end scheduling: how it works

In a nutshell, end-to-end scheduling in Tivoli Workload Scheduler enables you to schedule and control your jobs on the mainframe (Tivoli Workload Scheduler for z/OS), Windows, and UNIX environments for truly distributed scheduling. In the end-to-end configuration, Tivoli Workload Scheduler for z/OS is used as the planner for the job scheduling environment. Tivoli Workload Scheduler domain managers and fault-tolerant agents (FTAs) are used to schedule on the distributed platforms. So the agents then replace the use of tracker agents.

End-to-end scheduling directly connects Tivoli Workload Scheduler domain managers, and their underlying agents and domains, to Tivoli Workload Scheduler for z/OS. Tivoli Workload Scheduler for z/OS is thus seen as the master domain manager by the distributed network.

Tivoli Workload Scheduler for z/OS also creates the production scheduling plan for the distributed network and sends the plan to the domain managers. The domain managers then send a copy of the plan to each of their agents and subordinate managers for execution (Figure 13-8).

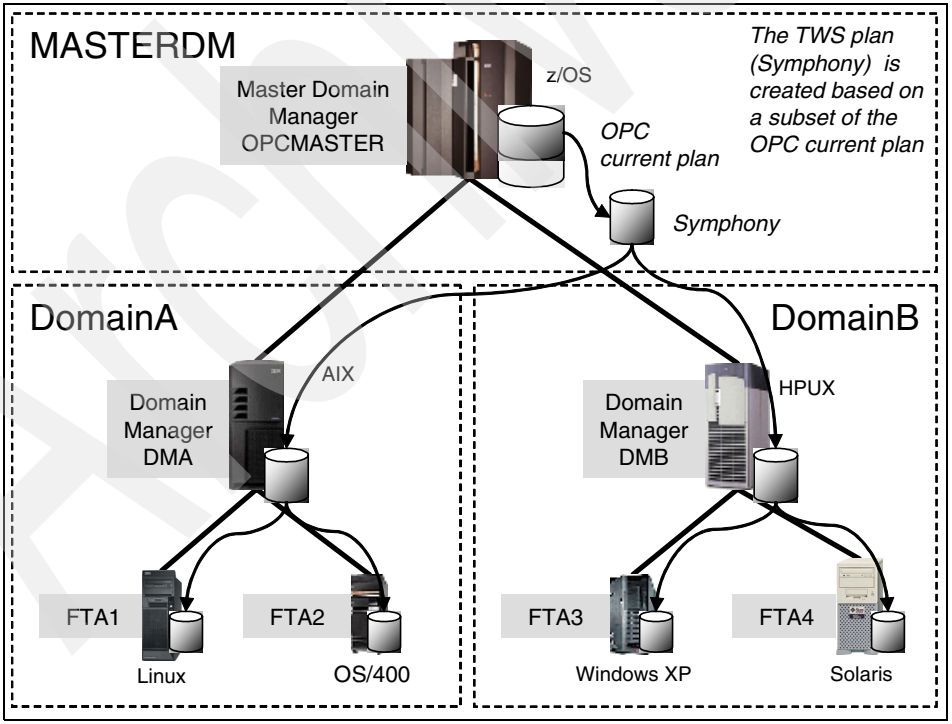


Figure 13-8 Tivoli Workload Scheduler for z/OS end-to-end plan distribution

Tivoli Workload Scheduler domain managers function as the broker systems for the distributed network by resolving all dependencies for all their subordinate managers and agents. They send their updates (in the form of events) to Tivoli Workload Scheduler for z/OS so that it can update the plan accordingly. Tivoli Workload Scheduler for z/OS handles its own jobs and notifies the domain managers of all the status changes of the Tivoli Workload Scheduler for z/OS jobs that involve the Tivoli Workload Scheduling plan. In this configuration, the domain managers and all distributed agents recognize Tivoli Workload Scheduler for z/OS as the master domain manager and notify it of all changes occurring in their own plans. At the same time, the agents are not permitted to interfere with the Tivoli Workload Scheduler for z/OS jobs, because they are viewed as running on the master that is the only node that is in charge of them.

Tivoli Workload Scheduler for z/OS also enables you to access job streams (schedules in Tivoli Workload Scheduler) and add them to the current plan in Tivoli Workload Scheduler for z/OS. You can also build dependencies among Tivoli Workload Scheduler for z/OS job streams and Tivoli Workload Scheduler jobs. From Tivoli Workload Scheduler for z/OS, you can monitor and control the distributed agents.

In the Tivoli Workload Scheduler for z/OS current plan, you can specify jobs to run on workstations in the Tivoli Workload Scheduler network. Tivoli Workload Scheduler for z/OS passes the job information to the Symphony file in the Tivoli Workload Scheduler for z/OS server, which in turn passes the Symphony file to the Tivoli Workload Scheduler domain managers (DMZ) to distribute and process. In turn, Tivoli Workload Scheduler reports the status of running and completed jobs back to the current plan for monitoring in the Tivoli Workload Scheduler for z/OS engine.

Table 13-1 shows the agents that can be used in an end-to-end environment. You should always check the Tivoli Workload Scheduler (Distributed) Release Notes for the latest information about the supported platforms and operating systems.

Table 13-1 List of agents that can be used in an end-to-end environment

Platform	Domain Manager	Fault-tolerant Agents
IBM AIX	X	X
HP-UX PA-RISC	X	X
Solaris Operating Environment	X	X
Microsoft® Windows NT®	X	X
Microsoft Windows 2000 and 2003 Server, Advanced Server	X	X

Platform	Domain Manager	Fault-tolerant Agents
Microsoft Windows 2000 Professional		X
Microsoft Windows XP Professional		X
Compaq Tru64		X
IBM OS/400		X
SGI Irix		X
IBM Sequent® Dynix		X
Red Hat Linux/INTEL	X	X
Red Hat Linux/390	X	X
Red Hat Linux/zSeries®		X
SUSE Linux/INTEL	X	X
SUSE Linux/390 and zSeries (kernel 2.4, 31-bits)	X	X
SUSE Linux/zSeries (kernel 2.4, 64-bits)		X
SUSE Linux/iSeries and pSeries® (kernel 2.4, 31-bits)		X

13.5 Comparing enterprise-wide scheduling deployment scenarios

In an environment with both mainframe and distributed scheduling requirements, in addition to end-to-end scheduling (managing both the mainframe and the distributed schedules from Tivoli Workload Scheduler for z/OS) there are two other alternatives, namely: Keeping Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS engines separate, or managing both mainframe and distributed environments from Tivoli Workload Scheduler (Distributed Scheduler), using the z/OS extended agent).

Note: Throughout this book, *end-to-end scheduling* refers to the type of environment where both the mainframe and the distributed schedules are managed from Tivoli Workload Scheduler for z/OS.

13.5.1 Keeping Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS separate

Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS can be used in conjunction with one another in an end-to-end environment, or they can be used separately. To keep them separate is purely up to you, it may be for specific business reasons or perhaps because separate people work directly with the UNIX or Windows systems than those who work on the mainframe. Whatever the case may be, the ability to keep Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS separate can be done.

Figure 13-9 on page 327 shows this type of environment.

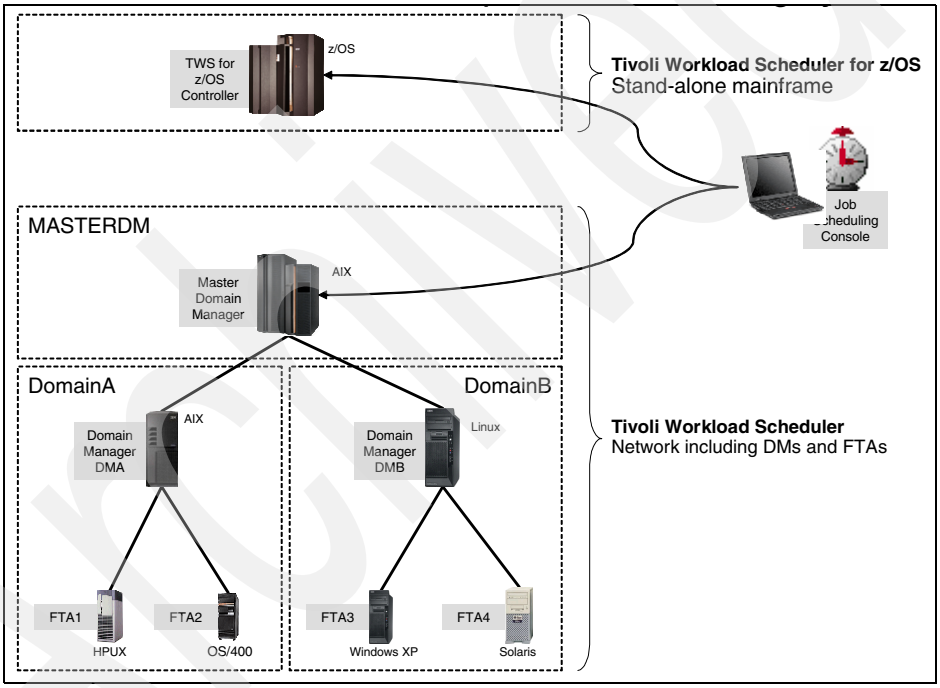


Figure 13-9 Keeping Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS separate

The results for keeping Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS separate is that your separate business groups can create their own planning, production, security, and distribution models and thus have their scheduling exist independently and continue to relate to the application requirements.

There are some operational and maintenance considerations for having two separate Tivoli Workload Scheduler planning engines:

- The dependencies between Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS have to be handled by the user. This can be done by using the extended agent on Tivoli Workload Scheduler, and outside the current scheduling dialogues by using dataset triggering or special resource flagging, both of which are available to Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS, to communicate between the two environments.
- It is difficult to manage the planning cycles of independent scheduling engines, especially since they do not have the same feature set. This makes it sometimes troublesome to meet the all dependency requirements between platforms.
- Keeping both Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS engines means there are two pieces of scheduling software to maintain.

Note: This scenario can be used as a bridge to end-to-end scheduling; that is, after deploying Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS products separately, you can connect the distributed environment to the mainframe and migrate the definitions to see end-to-end scheduling in action.

13.5.2 Managing both mainframe and distributed environments from Tivoli Workload Scheduler using the z/OS extended agent

It is also possible to manage both mainframe and distributed environments from Tivoli Workload Scheduler using the z/OS extended agent (Figure 13-10).

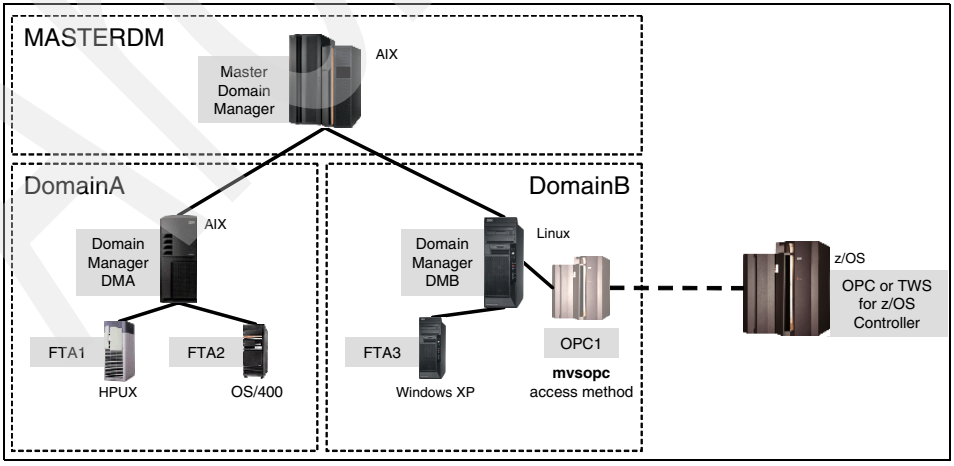


Figure 13-10 Managing both mainframe and distributed environments

This scenario has the following benefits:

- It is possible to do centralized monitoring and management. All scheduling status providing up-to-the-minute information about job and application states and current run-time statistics can be derived using the common user interface. Also you can define inter-platform dependencies for both scheduling environments.
- Each business unit can produce its own planning, production, security, and distribution model.
- Mainframe production planning and execution is handled separately from distributed job planning and execution. One does not (necessarily) affect the other.

Some of the considerations of an end-to-end environment are:

- The graphical interface shows distinct parts of the overall application flow, but not the overall picture. Also, there is no console (ISPF or Telnet) view that can show the entire plan end-to-end.
- The z/OS agent is another component that must be installed and maintained. Because the parallel engines can run on different cycles and there is no “master” coordinator to manage parallel tasks cross-platform, coordination of the engines needs careful planning.

13.5.3 Mainframe-centric configuration (or end-to-end scheduling)

This is the type of configuration that we cover in detail in this book. In this environment, Tivoli Workload Scheduler for z/OS is used to manage both the mainframe and the distributed schedules.

This scenario has the following benefits:

- All scheduling aspects from production planning, dependency resolution, and deadline management to job and script definition can be centrally managed.
- It has robust productive planning capabilities such as latest available job arrival times, critical job deadline times, and repeated applications.
- Localized job execution enables distributed execution to continue even during planned or unplanned system downtime.
- It allows console or GUI-based centralized monitoring. All scheduling status providing up-to-the-minute information about job and application states and current run-time statistics directly from the planning engine. There is also an alternate 3270-based single administrative console for enterprise scheduling.
- You can have enterprise application integration with this solution. Integration into both OS/390 and distributed-based applications including TBSM, SA/390, TDE, and WLM is possible.

Note: For more information about application integration in end-to-end environments, refer to *Integrating IBM Tivoli Workload Scheduler with Tivoli Products*, SG24-6648.

There are also some considerations, such as:

- Business units, e-business, and so forth are no longer autonomous. All management is done from the mainframe. No option exists to segregate components to be managed separately.
- There are a lot of moving parts, and proper planning, knowledge, and training are essential.

For more about benefits and considerations of the end-to-end scheduling environment, refer to 14.1.5, “Benefits of end-to-end scheduling” on page 357.

14

End-to-end scheduling architecture

End-to-end scheduling is an integrated solution for workload scheduling in an environment that includes both mainframe and non-mainframe systems.

In an end-to-end scheduling network, a mainframe computer acts as the single point of control for job scheduling across the entire enterprise. Tivoli Workload Scheduler for z/OS is used as the planner for the job scheduling environment. Tivoli Workload Scheduler fault-tolerant agents run work on the non-mainframe platforms, such as UNIX, Windows, and OS/400.

Because end-to-end scheduling involves running programs on multiple platforms, it is important to understand how the different components work together. We hope that this overview of end-to-end scheduling architecture will make it easier for you to install, use, and troubleshoot your system.

In this chapter, we introduce end-to-end scheduling and describe how it builds on the existing mainframe scheduling system, Tivoli Workload Scheduler for z/OS. If you are unfamiliar with Tivoli Workload Scheduler for z/OS, refer to the first part of the book (Part 1, “Tivoli Workload Scheduler for z/OS mainframe scheduling” on page 1) to get a better understanding of how the mainframe side of end-to-end scheduling works.

The following topics are covered in this chapter:

- End-to-end scheduling architecture
- Job Scheduling Console and related components
- Job log retrieval in an end-to-end environment
- Tivoli Workload Scheduler, important files, and directory structure
- conman commands in the end-to-end environment

14.1 End-to-end scheduling architecture

End-to-end scheduling means controlling scheduling from one end of an enterprise to the other—from the mainframe at the top of the network to the client workstations at the bottom. In the end-to-end scheduling solution, one or more Tivoli Workload Scheduler domain managers, and their underlying agents and domains, are put under the control of a Tivoli Workload Scheduler for z/OS engine. To the domain managers and FTAs in the network, the Tivoli Workload Scheduler for z/OS engine appears to be the master domain manager.

Tivoli Workload Scheduler for z/OS creates the plan (the Symphony file) for the entire end-to-end scheduling network. Tivoli Workload Scheduler for z/OS sends the plan down to the first-level domain managers. Each of these domain managers sends the plan to all of the subordinate workstations in its domain.

The domain managers act as brokers for the Tivoli Workload Scheduler network by resolving all dependencies for the subordinate workstations. They send their updates (in the form of events) to Tivoli Workload Scheduler for z/OS, which updates the plan accordingly. Tivoli Workload Scheduler for z/OS handles its own jobs and notifies the domain managers of all status changes of its jobs that involve the Tivoli Workload Scheduler plan. In this configuration, the domain manager and all Tivoli Workload Scheduler workstations recognize Tivoli Workload Scheduler for z/OS as the master domain manager and notify it of all of the changes occurring in their own plans. Tivoli Workload Scheduler workstations are not able to make changes to Tivoli Workload Scheduler for z/OS jobs.

Figure 14-1 on page 334 shows a Tivoli Workload Scheduler network managed by a Tivoli Workload Scheduler for z/OS engine. This is accomplished by connecting a Tivoli Workload Scheduler domain manager directly to the Tivoli Workload Scheduler for z/OS engine. The Tivoli Workload Scheduler for z/OS engine acts as the master domain manager of the Tivoli Workload Scheduler network.

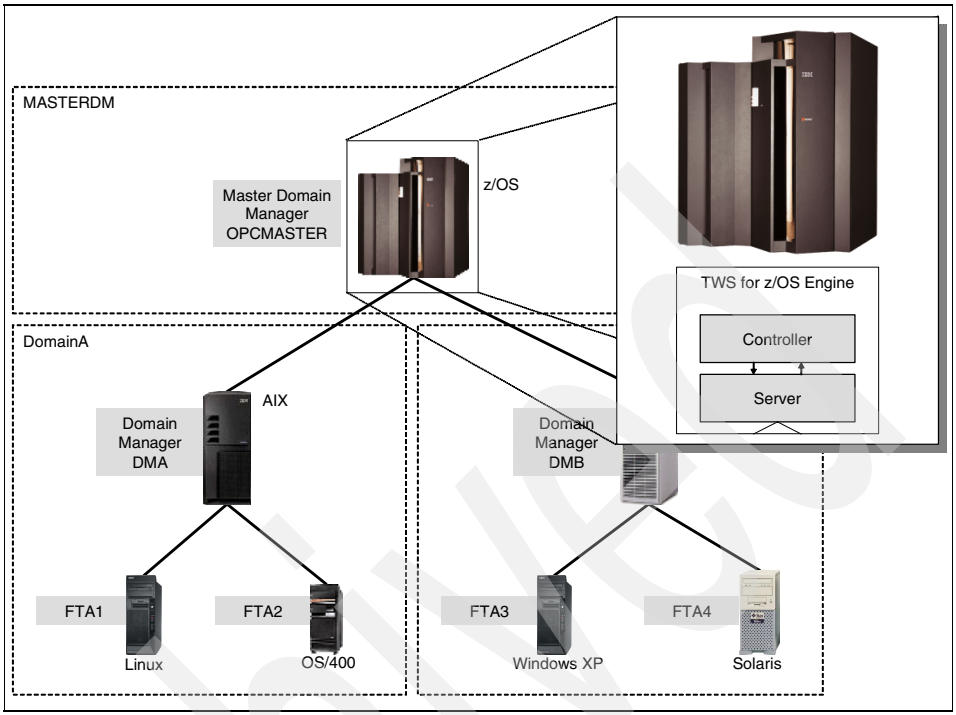


Figure 14-1 Tivoli Workload Scheduler for z/OS end-to-end scheduling

In Tivoli Workload Scheduler for z/OS, you can access job streams and add them to the current plan. In addition, you can create dependencies between Tivoli Workload Scheduler for z/OS jobs and Tivoli Workload Scheduler jobs. From Tivoli Workload Scheduler for z/OS, you can monitor and control all of the fault-tolerant agents in the network.

Note: Job streams are also known as *schedules* in Tivoli Workload Scheduler and *applications* in Tivoli Workload Scheduler for z/OS.

When you can specify that a job runs on a fault-tolerant agent, the Tivoli Workload Scheduler for z/OS engine includes the job information when the Symphony file is created on the mainframe. Tivoli Workload Scheduler for z/OS passes the Symphony file to the subordinate Tivoli Workload Scheduler domain managers, which then pass the file on to any subordinate DMs and FTAs. Tivoli Workload Scheduler on each workstation in the network reports the status of running and completed jobs back to the Tivoli Workload Scheduler for z/OS engine.

The Tivoli Workload Scheduler for z/OS engine is comprised of two components (started tasks on the mainframe): the controller and the server (also called the end-to-end server).

14.1.1 Components involved in end-to-end scheduling

To run the Tivoli Workload Scheduler for z/OS in an end-to-end configuration, you must have a Tivoli Workload Scheduler for z/OS server started task dedicated to end-to-end scheduling. This server started task is called the end-to-end server. The Tivoli Workload Scheduler for z/OS controller communicates with the FTAs using the end-to-end server, which starts several processes in z/OS UNIX System Services (USS). The processes running in USS use TCP/IP for communication with the subordinate FTAs.

The Tivoli Workload Scheduler for z/OS end-to-end server *must* run on the same z/OS systems where the Tivoli Workload Scheduler for z/OS controller runs.

Tivoli Workload Scheduler for z/OS end-to-end scheduling is comprised of three major components:

- The Tivoli Workload Scheduler for z/OS controller

  Manages database objects, creates plans with the workload, and executes and monitors the workload in the plan.

- The Tivoli Workload Scheduler for z/OS server

  Acts as the Tivoli Workload Scheduler master domain manager. It receives a part of the current plan from the Tivoli Workload Scheduler for z/OS controller. This plan contains job and job streams to be executed in the Tivoli Workload Scheduler network. The server is the focal point for all communication to and from the Tivoli Workload Scheduler network.

- Tivoli Workload Scheduler domain managers and fault-tolerant agents

  Domain managers serve as communication hubs between Tivoli Workload Scheduler for z/OS and the FTAs in each domain; fault-tolerant agents are usually where the majority of jobs are run.

Detailed description of the communication

Figure 14-2 on page 336 shows the communication between the Tivoli Workload Scheduler for z/OS controller and the Tivoli Workload Scheduler for z/OS server.

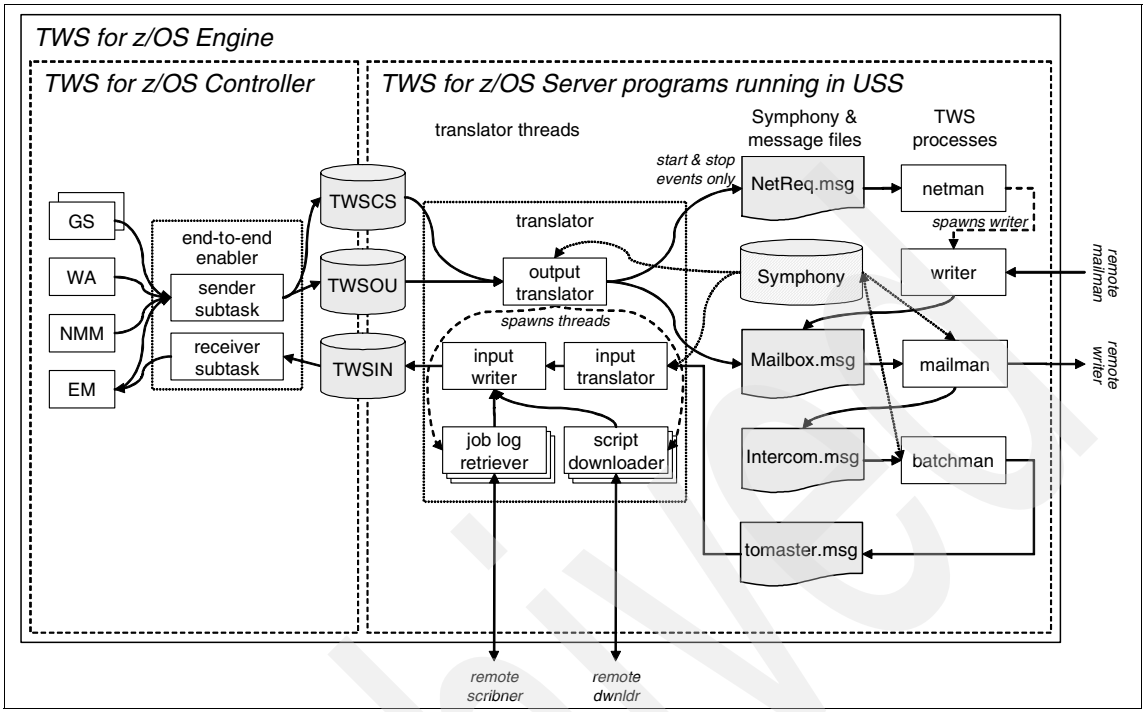


Figure 14-2 Tivoli Workload Scheduler for z/OS 8.2 interprocess communication

Tivoli Workload Scheduler for z/OS server processes and tasks

The end-to-end server address space hosts the tasks and the data sets that function as the intermediaries between the controller and the subordinate domain managers. Most of these programs and files have equivalents on Tivoli Workload Scheduler workstations.

The Tivoli Workload Scheduler for z/OS server uses the following processes, threads, and tasks (see Figure 14-2):

netman The Tivoli Workload Scheduler network listener daemon. It is started automatically when the end-to-end server task starts. The netman process monitors the NetReq.msg queue and listens to the TCP port defined in the server topology portnumber parameter. (Default is port 31111.) When netman receives a request, it starts another program to handle the request, usually writer or mailman. Requests to start or stop mailman are written by output translator to the NetReq.msg queue. Requests to start or stop writer are sent via TCP by the mailman process on a remote workstation (domain manager at the first level).

writer One writer process is started by netman for each remote workstation that has established an uplink to the Tivoli Workload Scheduler for z/OS end-to-end server. Each writer process receives events from the mailman process running on the remote workstation and writes these events to the Mailbox.msg file.

mailman The main message handler process. Its main tasks are:

- Routing events. It reads the events stored in the Mailbox.msg queue and sends them either to the controller (writing them in the Intercom.msg file) or to the writer process on a remote workstation (via TCP).
- Linking to remote workstations (domain managers at the first level). The mailman process requests that the netman program on each remote workstation starts a writer process to accept the connection.
- Sending the Symphony file to subordinate workstations (domain managers at the first level). When a new Symphony file is created, the mailman process sends a copy of the file to each subordinate domain manager and fault-tolerant agent.

batchman Updates the Symphony file and resolves dependencies at the level of the master domain manager. After the Symphony file has been written the first time, batchman is the only program that makes changes to the file.

Important: No jobman process runs in UNIX System Services. Mainframe jobs, including any that should be run in USS, must be submitted in Tivoli Workload Scheduler for z/OS on a normal (non-FTA) CPU workstation.

translator Through its input and output threads (discussed in more detail later), the translator process translates events from Tivoli Workload Scheduler format to Tivoli Workload Scheduler for z/OS format and vice versa. The translator program was developed specifically for the end-to-end scheduling solution. The translator process runs only in UNIX System Services on the mainframe; it does not run on ordinary Tivoli Workload Scheduler workstations such as domain managers and FTAs. The translator program provides the glue that binds Tivoli Workload Scheduler for z/OS and Tivoli Workload Scheduler by enabling these two products to function as a unified scheduling system.

job log retriever A thread of the translator process that is spawned to fetch a job log from a fault-tolerant agent. One job log retriever thread is spawned for each requested FTA job log.

The job log retriever receives the log, sizes it according to the LOGLINES parameter, translates it from UTF-8 to EBCDIC, and queues it in the *inbound queue* of the controller. The retrieval of a job log is a lengthy operation and can take a few moments to complete.

The user may request several logs at the same time. The job log retriever thread terminates after the log has been written to the inbound queue. If using the Tivoli Workload Scheduler for z/OS ISPF panel interface, the user will be notified by a message when the job log has been received.

script downloader A thread of the translator process that is spawned to download the script for an operation (job) whoseTivoli Workload Scheduler Centralized Script option is set to Yes. One script downloader thread is spawned for each script that must be downloaded. Several script downloader threads can be active at the same time. The script that is to be downloaded is received from the output translator.

starter The first process that is started in UNIX System Services when the end-to-end server started task is started. The starter process (not shown in Figure 14-2 on page 336) starts the translator and netman processes.

These events are passed from the server to the controller:

input translator A thread of the translator process. The input translator thread reads events from the tomaster.msg file and translates them from Tivoli Workload Scheduler format to Tivoli Workload Scheduler for z/OS format. It also performs UTF-8 to EBCDIC translation and sends the translated events to the input writer.

input writer Receives the input from the job log retriever, input translator, and script downloader and writes it in the inbound queue (the EQQTWSIN data set).

receiver subtask A subtask of the end-to-end task run in the Tivoli Workload Scheduler for z/OS controller. Receives events from the inbound queue and queues them to the Event Manager task.

These events are passed from the controller to the server:

sender subtask A subtask of the end-to-end task in the Tivoli Workload Scheduler for z/OS controller. Receives events for changes to the current plan that are related to Tivoli Workload Scheduler fault-tolerant agents. The Tivoli Workload Scheduler for z/OS tasks that can change the current plan are: General Service (GS), Normal Mode Manager (NMM), Event Manager (EM), and Workstation Analyzer (WA).

The events are communicated via SSI; this is the method used by Tivoli Workload Scheduler for z/OS tasks to exchange events.

The NMM sends synchronization events to the sender task whenever the plan is extended, replanned, or refreshed, and any time the Symphony file is renewed.

output translator A thread of the translator process. The output translator thread reads events from the *outbound* queue. It translates the events from Tivoli Workload Scheduler for z/OS format to Tivoli Workload Scheduler format and evaluates them, performing the appropriate function. Most events, including those related to changes to the Symphony file, are written to Mailbox.msg. Requests to start or stop netman or mailman are written to NetReq.msg. Output translator also translates events from EBCDIC to UTF-8.

The output translator performs different actions, depending on the type of the event:

- Starts a job log retriever thread if the event is to retrieve the log of a job from an FTA.
- Starts a script downloader thread if the event is to send a script to an FTA.
- Queues an event in NetReq.msg if the event is to start or stop mailman.
- Queues events in Mailbox.msg for the other events that are sent to update the Symphony file on the FTAs. Examples include events for a change of job status, events for manual changes on jobs or workstations, and events to link and unlink workstations.
- Switches the Symphony files.

Tivoli Workload Scheduler for z/OS data sets and files used for end-to-end scheduling

The Tivoli Workload Scheduler for z/OS server and controller use the following data sets and files:

EQQTWSIN The inbound queue. Sequential data set used to queue events sent by the server *into* the controller. Must be defined in Tivoli Workload Scheduler for z/OS controller and the end-to-end server started task procedure (shown as `TWSIN` in Figure 14-2 on page 336).

EQQTWSOU The outbound queue. Sequential data set used to queue events sent by the controller *out* to the server. Must be defined in Tivoli Workload Scheduler for z/OS controller and the end-to-end server started task procedure (shown as `TWSOU` in Figure 14-2 on page 336).

EQQTWSCS Partitioned data set used temporarily to store scripts that are in the process of being sent to an FTA immediately prior to submission of the corresponding job. The sender subtask copies the script to a new member in this data set from the JOBLIB data set. This data set is shown as `TWSCS` in Figure 14-2 on page 336.

This data set is described in "Tivoli Workload Scheduler for z/OS end-to-end database objects" on page 342. It is not shown in Figure 14-2 on page 336.

Symphony HFS file containing the active copy of the plan used by the Tivoli Workload Scheduler fault-tolerant agents.

Sinfonia HFS file containing the copy of the plan that is distributed to the fault-tolerant agents. This file is not shown in Figure 14-2 on page 336.

NetReq.msg HFS file used to queue requests for the netman process.

Mailbox.msg HFS file used to queue events sent to the mailman process.

intercom.msg HFS file used to queue events sent to the batchman process.

tomaster.msg HFS file used to queue events sent to the input translator process.

Translator.chk HFS file used as checkpoint file for the translator process. It is equivalent to the checkpoint data set used by the Tivoli Workload Scheduler for z/OS controller. For example, it contains information about the status of the Tivoli

Workload Scheduler for z/OS current plan, Symphony run number, Symphony availability. This file is not shown in Figure 14-2 on page 336.

Translator.wjl HFS file used to store information about job log retrieval and script downloading that are in progress. At initialization, the translator checks the translator.wjl file for job log retrievals and script downloads that did not complete (either successfully or in error) and sends the error back to the controller. This file is not shown in Figure 14-2 on page 336.

EQQSCLIB Partitioned data set used as a repository for jobs with non-centralized script definitions running on FTAs. The EQQSCLIB data set is described in "Tivoli Workload Scheduler for z/OS end-to-end database objects" on page 342. It is not shown in Figure 14-2 on page 336.

EQQSCPDS VSAM data sets containing a copy of the current plan used by the daily plan batch programs to create the Symphony file.

The end-to-end plan-creating process is described in 14.1.3, "Tivoli Workload Scheduler for z/OS end-to-end plans" on page 348. It is not shown in Figure 14-2 on page 336.

14.1.2 Tivoli Workload Scheduler for z/OS end-to-end configuration

The topology of the Tivoli Workload Scheduler network that is connected to the Tivoli Workload Scheduler for z/OS engine is described in parameter statements for the Tivoli Workload Scheduler for z/OS server and for the Tivoli Workload Scheduler for z/OS programs that handle the long-term plan and the current plan.

Parameter statements are also used to activate the end-to-end subtasks in the Tivoli Workload Scheduler for z/OS controller.

The parameter statements that are used to describe the topology are covered in 15.1.6, "Initialization statements for Tivoli Workload Scheduler for z/OS end-to-end scheduling" on page 394. This section also includes an example of how to reflect a specific Tivoli Workload Scheduler network topology in Tivoli Workload Scheduler for z/OS servers and plan programs using the Tivoli Workload Scheduler for z/OS topology parameter statements.

Tivoli Workload Scheduler for z/OS end-to-end database objects

In order to run jobs on fault-tolerant agents or extended agents, you must first define database objects related to the Tivoli Workload Scheduler workload in Tivoli Workload Scheduler for z/OS databases.

The Tivoli Workload Scheduler for z/OS end-to-end related database objects are:

- Fault-tolerant workstations

  A fault-tolerant workstation is a computer workstation configured to schedule jobs on FTAs. *The workstation must also be defined in the server CPUREC initialization statement* (Figure 14-3).

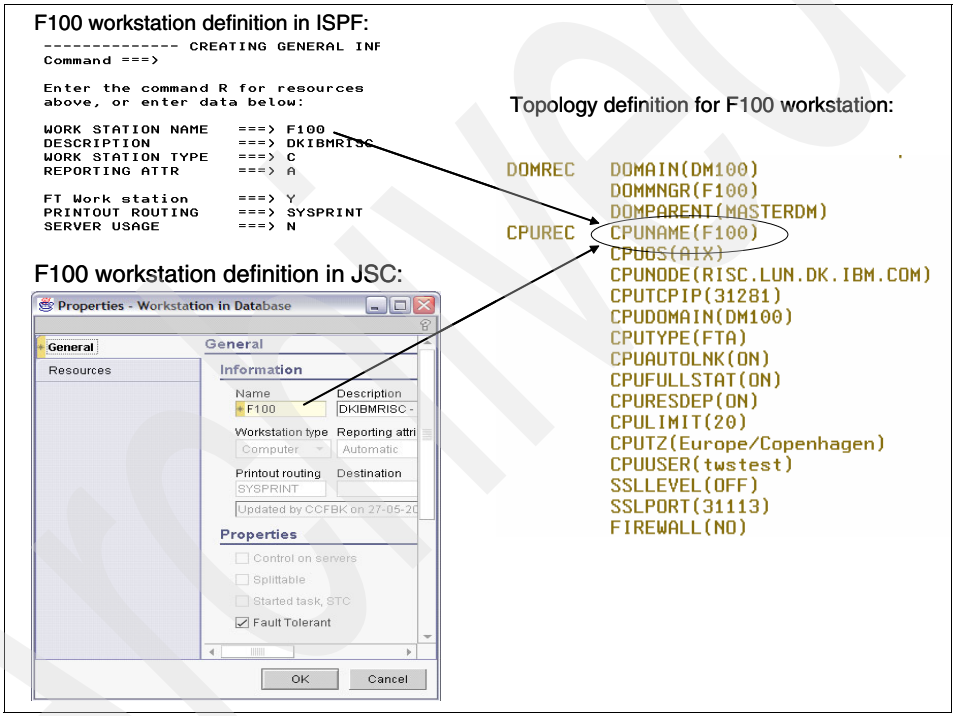


Figure 14-3 A workstation definition and its corresponding CPUREC

- Job streams, jobs, and their dependencies

  Job streams and jobs that are intended to be run on FTAs are defined like other job streams and jobs in Tivoli Workload Scheduler for z/OS. To run a job on a Tivoli Workload Scheduler FTA, the job is simply defined on a fault-tolerant workstation. Dependencies between FTA jobs are created exactly the same way as other job dependencies in the Tivoli Workload Scheduler for z/OS controller. This is also the case when creating dependencies between FTA jobs and mainframe jobs.

Some of the Tivoli Workload Scheduler for z/OS mainframe-specific options are not available for FTA jobs.

- Tivoli Workload Scheduler for z/OS resources

  In an end-to-end scheduling network, all resource dependencies are global. This means that the resource dependency is resolved by the Tivoli Workload Scheduler for z/OS controller and not locally on the FTA.

  For a job running on an FTA, the use of resources entails a loss of fault tolerance. Only the controller determines the availability of a resource and consequently lets the FTA start the job. Thus, if a job running on an FTA uses a resource, the following occurs:

  – When the resource is available, the controller sets the state of the job to started and the extended status to waiting for submission.
  – The controller sends a release-dependency event to the FTA.
  – The FTA starts the job.

  If the connection between the engine and the FTA is down, the operation will not start on the FTA even if the resource is available. The operation will start only after the connection has been re-established.

Note: Special Resource dependencies are represented differently depending on whether you are looking at the job through Tivoli Workload Scheduler for z/OS interfaces or Tivoli Workload Scheduler interfaces. If you observe the job using Tivoli Workload Scheduler for z/OS interfaces, you can see the resource dependencies as expected.

However, when you monitor a job on a fault-tolerant agent by means of the Tivoli Workload Scheduler interfaces, you will not be able to see the resource that is used by the job. Instead you will see a dependency on a job called `OPCMASTER#GLOBAL.SPECIAL_RESOURCES`. This dependency is set by the engine. Every job that has special resource dependencies has a dependency to this job.

When the engine allocates the resource for the job, the dependency is released. (The engine sends a release event for the specific job through the network.)

- The task or script associated with the FTA job, defined in Tivoli Workload Scheduler for z/OS

  In Tivoli Workload Scheduler for z/OS 8.2, the task or script associated to the FTA job can be defined in two different ways:

  a. Non-centralized script

     Defined in a special partitioned data set, EQQSCLIB, allocated in the Tivoli Workload Scheduler for z/OS controller started task procedure, stores the job or task definitions for FTA jobs. The script (the JCL) resides on the fault-tolerant agent. This is the default behavior in Tivoli Workload Scheduler for z/OS for fault-tolerant agent jobs.

  b. Centralized script

     Defines the job in Tivoli Workload Scheduler for z/OS with the Centralized Script option set to Y (Yes).

     Note: The default for all operations and jobs in Tivoli Workload Scheduler for z/OS is N (No).

     A centralized script resides in the Tivoli Workload Scheduler for z/OS JOBLIB and is downloaded to the fault-tolerant agent every time the job is submitted. The concept of centralized scripts has been added for compatibility with the way that Tivoli Workload Scheduler for z/OS manages jobs in the z/OS environment.

Non-centralized script

For every FTA job definition in Tivoli Workload Scheduler for z/OS where the Centralized Script option is set to N (non-centralized script) there must be a corresponding member in the EQQSCLIB data set. The members of EQQSCLIB contain a JOBREC statement that describes the path to the job or the command to be executed and eventually the user to be used when the job or command is executed.

Example for a UNIX script:

```
JOBREC JOBSCR(/Tivoli/tws/scripts/script001_accounting)
JOBUSR(userid01)
```

Example for a UNIX command:

```
JOBREC JOBCMD(ls) JOBUSR(userid01)
```

If the JOBUSR (user for the job) keyword is not specified, the user defined in the CPUUSER keyword of the CPUREC statement for the fault-tolerant workstation is used.

If necessary, Tivoli Workload Scheduler for z/OS JCL variables can be used in the JOBREC definition. Tivoli Workload Scheduler for z/OS JCL variables and variable substitution in a EQQSCLIB member is managed and controlled by VARSUB statements placed directly in the EQQSCLIB member with the JOBREC definition for the particular job.

Furthermore, it is possible to define Tivoli Workload Scheduler recovery options for the job defined in the JOBREC statement. Tivoli Workload Scheduler recovery options are defined with RECOVERY statements placed directly in the EQQSCLIB member with the JOBREC definition for the particular job.

The JOBREC (and optionally VARSUB and RECOVERY) definitions are read by the Tivoli Workload Scheduler for z/OS plan programs when producing the new current plan and placed as part of the job definition in the Symphony file.

If an FTA job stream is added to the plan in Tivoli Workload Scheduler for z/OS, the JOBREC definition will be read by Tivoli Workload Scheduler for z/OS, copied to the Symphony file on the Tivoli Workload Scheduler for z/OS server, and sent (as events) by the server to the fault-tolerant agent Symphony files via any domain managers that lie between the FTA and the mainframe.

It is important to remember that the EQQSCLIB member only has a pointer (the path) to the job that is going to be executed. The actual job (the JCL) is placed locally on the FTA or workstation in the directory defined by the JOBREC JOBSCR definition.

This also means that it is not possible to use the JCL edit function in Tivoli Workload Scheduler for z/OS to edit the script (the JCL) for jobs where the script (the pointer) is defined by a JOBREC statement in the EQQSCLIB data set.

Centralized script

The script for a job defined with the Centralized Script option set to Y must be defined in Tivoli Workload Scheduler for z/OS JOBLIB. The script is defined the same way as normal JCL.

It is possible (but not necessary) to define some parameters of the centralized script, such as the user, in a job definition member of the SCRPTLIB data set.

With centralized scripts, you can perform variable substitution, automatic recovery, JCL editing, and job setup (as for “normal” z/OS jobs defined in the Tivoli Workload Scheduler for z/OS JOBLIB). It is also possible to use the job-submit exit (EQQUX001).

Note that jobs with a centralized script will be defined in the Symphony file with a dependency named `script`. This dependency will be released when the job is

ready to run *and* the script is downloaded from the Tivoli Workload Scheduler for z/OS controller to the fault-tolerant agent.

To download a centralized script, the DD statement EQQTWSCS must be present in the controller and server started tasks. During the download the <twshome>/centralized directory is created at the fault-tolerant workstation. The script is downloaded to this directory. If an error occurs during this operation, the controller retries the download every 30 seconds for a maximum of 10 times. If the script download still fails after 10 retries, the job (operation) is marked as `Ended-in-error` with error code `OSUF`.

Here are the detailed steps for downloading and executing centralized scripts on FTAs (Figure 14-4 on page 347):

1. Tivoli Workload Scheduler for z/OS controller instructs sender subtask to begin script download.
2. The sender subtask writes the centralized script to the centralized scripts data set (EQQTWSCS).
3. The sender subtask writes a script download event (type JCL, action D) to the output queue (EQQTWSOU).
4. The output translator thread reads the JCL-D event from the output queue.
5. The output translator thread reads the script from the centralized scripts data set (EQQTWSCS).
6. The output translator thread spawns a script downloader thread.
7. The script downloader thread connects directly to netman on the FTA where the script will run.
8. netman spawns dwnldr and connects the socket from the script downloader thread to the new dwnldr process.
9. dwnldr downloads the script from the script downloader thread and writes it to the TWSHome/centralized directory on the FTA.
10. dwnldr notifies the script downloader thread of the result of the download.
11. The script downloader thread passes the result to the input writer thread.
12. If the script download was successful, the input writer thread writes a script download successful event (type JCL, action C) on the input queue (EQQTWSIN). If the script download was unsuccessful, the input writer thread writes a script download in error event (type JCL, action E) on the input queue.
13. The receiver subtask reads the script download result event from the input queue.

14. The receiver subtask notifies the Tivoli Workload Scheduler for z/OS controller of the result of the script download. If the result of the script download was successful, the OPC controller then sends a release dependency event (type JCL, action R) to the FTA, via the normal IPC channel (sender subtask → output queue → output translator → Mailbox.msg → mailman → writer on FTA, and so on). This event causes the job to run.

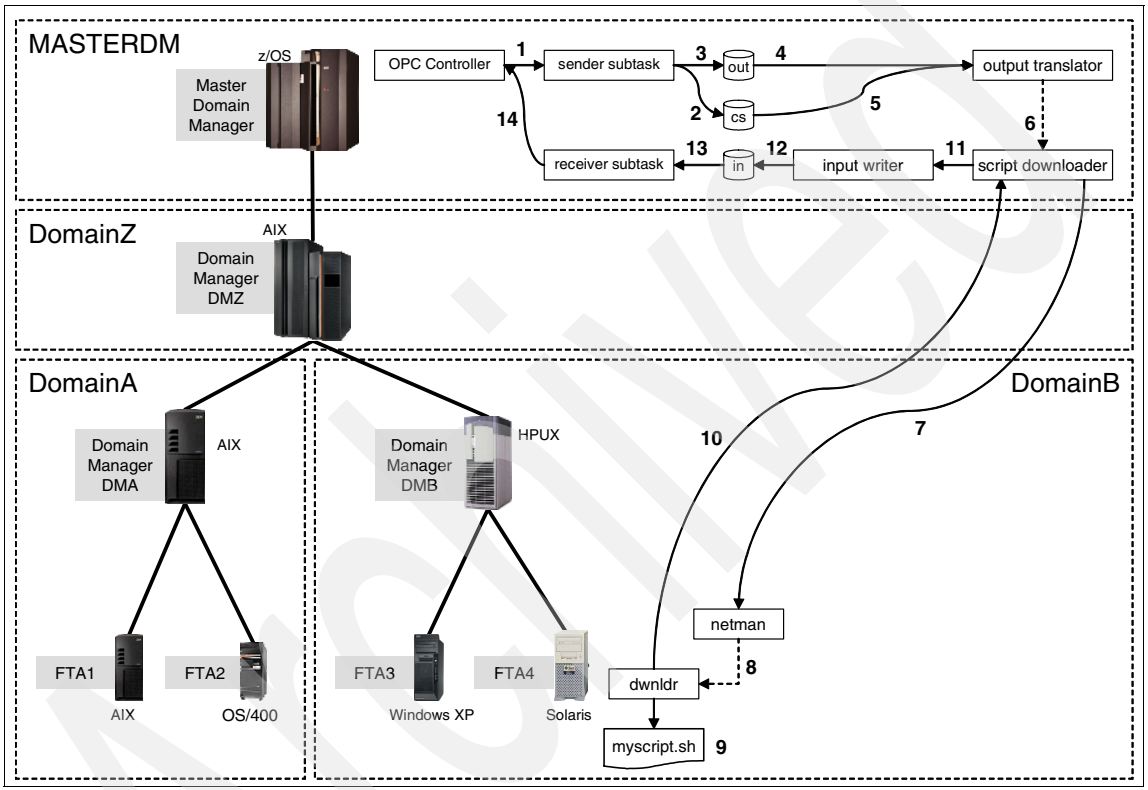


Figure 14-4 Steps and processes for downloading centralized scripts

Creating a centralized script in the Tivoli Workload Scheduler for z/OS JOBLIB data set is described in 15.4.2, “Definition of centralized scripts” on page 452.

14.1.3 Tivoli Workload Scheduler for z/OS end-to-end plans

When the end-to-end enabler is installed and configured, at least one fault-tolerant agent workstation is defined, and at least one FTA job is defined, a new Symphony file will be built automatically each time the Tivoli Workload Scheduler for z/OS current plan program is run. This program runs whenever the current plan is extended, refreshed, or replanned. The Symphony file is the subset of the Tivoli Workload Scheduler for z/OS current plan that includes work for fault-tolerant agents.

The Tivoli Workload Scheduler for z/OS current plan is normally extended on workdays. Figure 14-5 shows a combined view of long-term planning and current planning. Changes to the databases require an update of the long-term plan, thus most sites run the LTP Modify batch job immediately before extending the current plan.

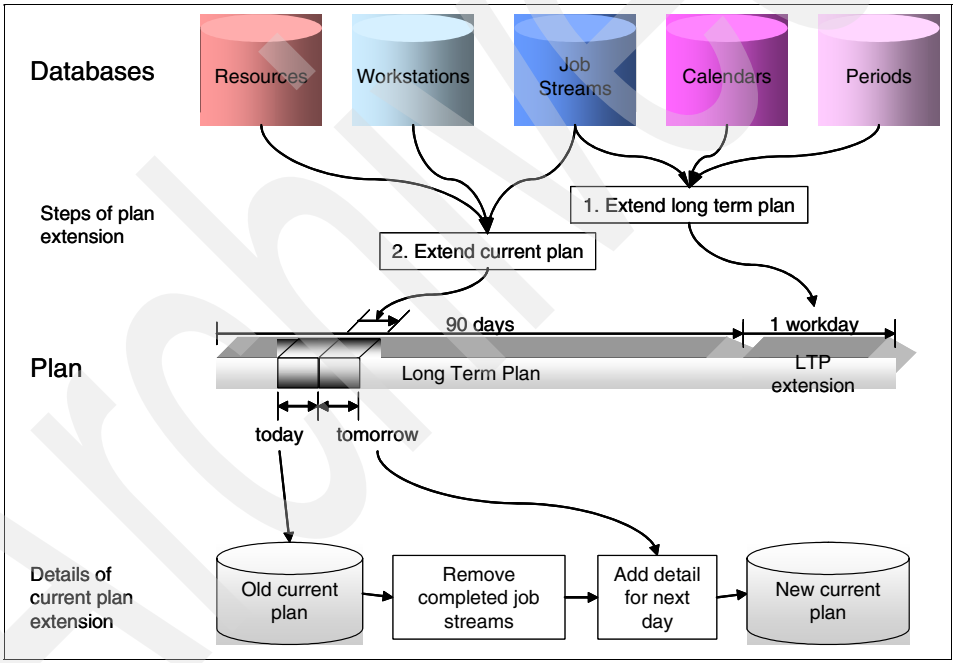


Figure 14-5 Combined view of the long-term planning and current planning

If the end-to-end feature is activated in Tivoli Workload Scheduler for z/OS, the current plan program will read the topology definitions described in the TOPLOGY, DOMREC, CPUREC, and USRREC initialization statements (see 14.1.2, “Tivoli Workload Scheduler for z/OS end-to-end configuration” on page 341) and the script library (EQQSCLIB) as part of the planning process. Information from the initialization statements and the script library will be used to create a Symphony

file for the Tivoli Workload Scheduler FTAs (Figure 14-6). Tivoli Workload Scheduler for z/OS planning programs handle the whole process, as described in the next section.

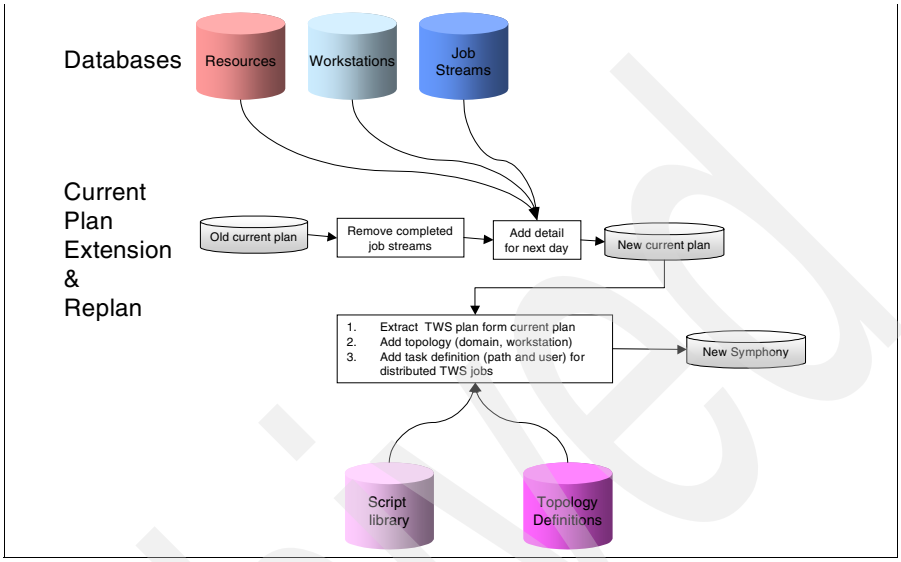


Figure 14-6 Creating Symphony file in Tivoli Workload Scheduler for z/OS plan programs

Detailed description of the Symphony creation

Figure 14-2 on page 336 gives a description of the tasks and processes involved in the Symphony creation.

1. The process is handled by Tivoli Workload Scheduler for z/OS planning batch programs. The batch produces the NCP and initializes the symUSER.
2. The Normal Node Manager (NMM) sends the SYNC START ('S') event to the server, and the E2E receiver starts, leaving all events in the inbound queue (TWSIN).
3. When the SYNC START ('S') is processed by the output translator, it stops the OPCMASTER, sends the SYNC END ('E') to the controller, and stops the entire network.
4. The NMM applies the job-tracking events received while the new plan was produced. It then copies the new current plan data set (NCP) to the Tivoli Workload Scheduler for z/OS current plan data set (CP1 or CP2), makes a current plan backup (copies active CP1/CP2 to inactive CP1/CP2), and creates the Symphony Current Plan (SCP) data set as a copy of the active current plan (CP1 or CP2) data set.
5. Tivoli Workload Scheduler for z/OS mainframe schedule is resumed.

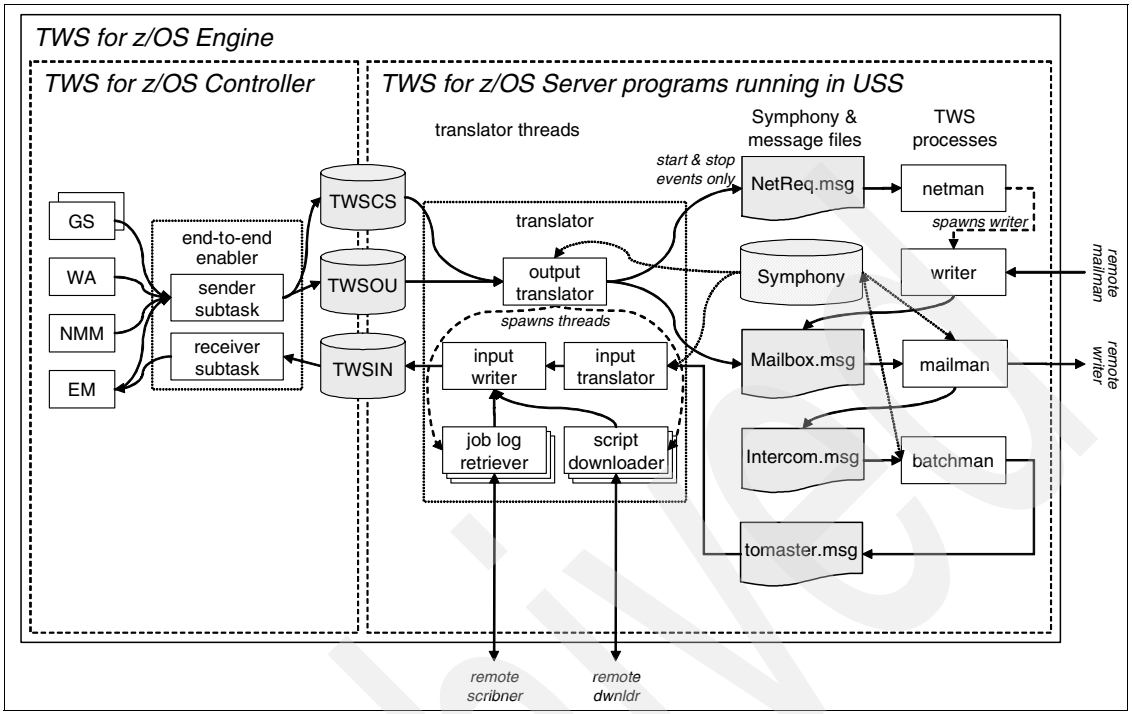


Figure 14-7 Tivoli Workload Scheduler for z/OS 8.2 interprocess communication

6. The end-to-end receiver begins to process events in the queue.
7. The SYNC CPREADY ('Y') is sent to the output translator and starts, leaving all of the events in the outbound queue (TWSOU).
8. The plan program starts producing the SymUSER file starting from SCP and then renames it Symnew.
9. When the Symnew file has been created, the plan program ends and NMM notifies the output translator that the Symnew file is ready, sending the SYNC SYMREADY ('R') event to the output translator.
10. The output translator renames old Symphony and Sinfonia files to Symold and Sinfold files, and a Symphony OK ('X') or NOT OK ('B') Sync event is sent to the Tivoli Workload Scheduler for z/OS engine, which logs a message in the engine message log indicating whether the Symphony has been switched.
11. The Tivoli Workload Scheduler for z/OS server master is started in USS and the Input Translator starts to process new events. As in Tivoli Workload Scheduler, mailman and batchman process events are left in local event files and start distributing the new Symphony file to the whole Tivoli Workload Scheduler network.

When the Symphony file is created by the Tivoli Workload Scheduler for z/OS plan programs, it (or, more precisely, the Sinfonia file) will be distributed to the Tivoli Workload Scheduler for z/OS subordinate domain manager, which in turn distributes the Symphony (Sinfonia) file to its subordinate domain managers and fault-tolerant agents (Figure 14-8).

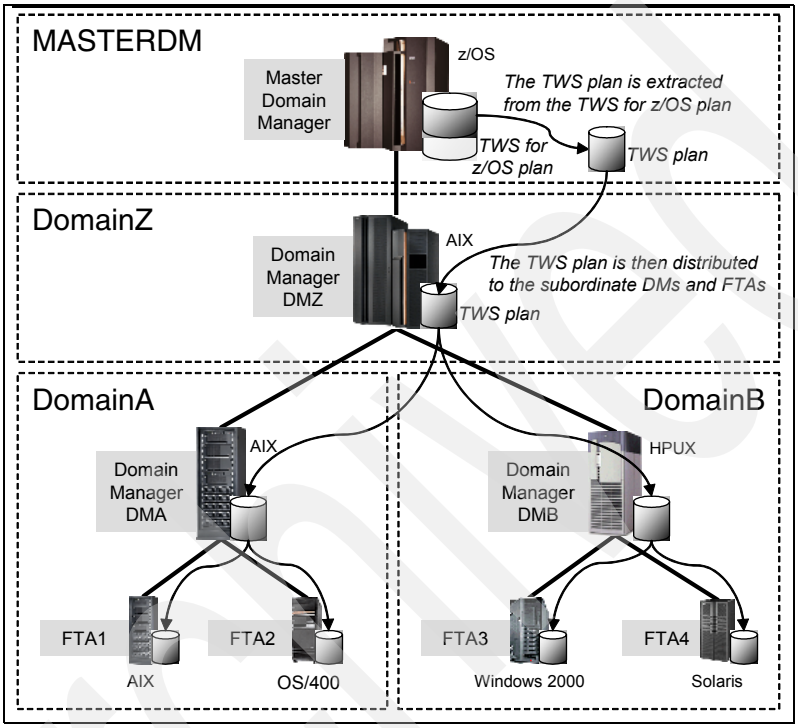


Figure 14-8 Symphony file distribution to FTWs

The Symphony file is generated:

- Every time the Tivoli Workload Scheduler for z/OS plan is extended or replanned
- When a Symphony renew batch job is submitted (from Tivoli Workload Scheduler for z/OS ISPF panels, option 3.5)

The Symphony file contains:

- Jobs to be executed on Tivoli Workload Scheduler FTAs
- z/OS (mainframe) jobs that are predecessor FTA jobs
- Job streams that have at least one job in the Symphony file

- Topology information for the Tivoli Workload Scheduler network with all workstation and domain definitions, including the master domain manager of the Tivoli Workload Scheduler network; that is, the Tivoli Workload Scheduler for z/OS host.

After the Symphony file is created and distributed to the Tivoli Workload Scheduler FTAs, the Symphony file is updated by events:

- When job status changes
- When jobs or job streams are modified
- When jobs or job streams for the Tivoli Workload Scheduler FTAs are added to the plan in the Tivoli Workload Scheduler for z/OS controller.

If you look at the Symphony file locally on a Tivoli Workload Scheduler FTA, from the Job Scheduling Console, or using the Tivoli Workload Scheduler command line interface to the plan (conman), you will see that:

- The Tivoli Workload Scheduler workstation has the same name as the related workstation defined in Tivoli Workload Scheduler for z/OS for the agent. OPCMASTER is the hard-coded name for the master domain manager workstation for the Tivoli Workload Scheduler for z/OS controller.
- The name of the job stream (or schedule) is the hexadecimal representation of the occurrence token, a unique identifier of the job stream instance. The job streams are always associated with a workstation called OPCMASTER. The OPCMASTER workstation is essentially the parts of the Tivoli Workload Scheduler for z/OS end-to-end server that run in UNIX System Services (Figure 14-9 on page 353).
- Using the occurrence token as the name of the job stream instance makes it possible to have several instances for the same job stream in the plan at the same time. This is important because in the Tivoli Workload Scheduler Symphony file, the job stream name is used as the unique identifier. Moreover, it is possible to have a plan in the Tivoli Workload Scheduler for z/OS controller and a Symphony file that spans more than 24 hours.

Note: In the Tivoli Workload Scheduler for z/OS plan, the key (unique identifier) for a job stream occurrence is job stream name and input arrival time.

In the Tivoli Workload Scheduler Symphony file, the key is the job stream instance name. Because Tivoli Workload Scheduler for z/OS can have several job stream instances with the same name in the plan, it is necessary with an unique and invariant identifier (the occurrence token) for the occurrence or job stream instance name in the Symphony file.

- The job name is made up using one of the following formats (see Figure 14-9 on page 353 for an example):
  - `<T>_<opnum>_<applname>`
    when the job is created in the Symphony file
  - `<T>_<opnum>_<ext>_<applname>`
    when the job is first deleted from the current plan and then re-created in the current plan

  In these examples:

  - `<T>` is `J` for normal jobs (operations), `P` for jobs that are representing pending predecessors, or `R` for recovery jobs (jobs added by Tivoli Workload Scheduler recovery).
  - `<opnum>` is the operation number for the job in the job stream (in the current plan).
  - `<ext>` is a sequential number that is incremented every time the same operation is deleted then re-created in the current plan; if `0`, it is omitted.
  - `<applname>` is the name of the occurrence (job stream) the operation belongs to.

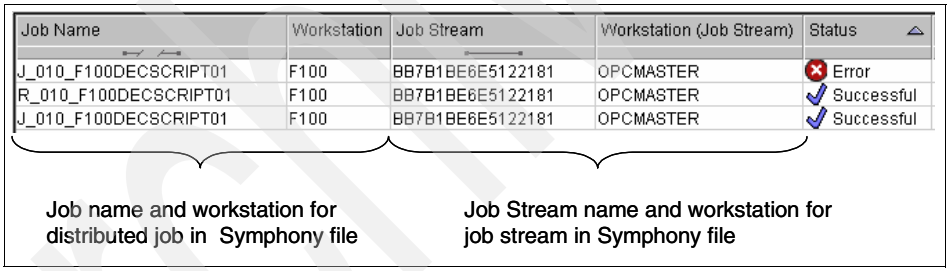


Figure 14-9 Job name and job stream name as generated in the Symphony file

Tivoli Workload Scheduler for z/OS uses the job name and an operation number as “key” for the job in a job stream.

In the Symphony file only the job name is used as key. Tivoli Workload Scheduler for z/OS can have the same job name several times in on job stream and distinguishes between identical job names with the operation number, so the job names generated in the Symphony file contain the Tivoli Workload Scheduler for z/OS operation number as part of the job name.

The name of a job stream (application) can contain national characters such as dollar ($), sect (§), and pound (£). These characters are converted into dashes (-) in the names of included jobs when the job stream is added to the Symphony file

or when the Symphony file is created. For example, consider the job stream name:

```
APPL$$234§§ABC£
```

In the Symphony file, the names of the jobs in this job stream will be:

```
<T>_<opnum>_APPL--234--ABC-
```

This nomenclature is still valid because the job stream instance (occurrence) is identified by the occurrence token, and the operations are each identified by the operation numbers (<opnum>) that are part of the job names in the Symphony file.

Note: The criteria that are used to generate job names in the Symphony file can be managed by the Tivoli Workload Scheduler for z/OS JTOPTS TWSJOBNAME() parameter, which was introduced with APAR PQ77970. It is possible, for example, to use the job name (from the operation) instead of the job stream name for the job name in the Symphony file, so the job name will be <T>_<opnum>_<jobname> in the Symphony file.

In normal situations, the Symphony file is automatically generated as part of the Tivoli Workload Scheduler for z/OS plan process. The topology definitions are read and built into the Symphony file as part of the Tivoli Workload Scheduler for z/OS plan programs, so regular operation situations can occur where you need to renew (or rebuild) the Symphony file from the Tivoli Workload Scheduler for z/OS plan:

- When you make changes to the script library or to the definitions of the TOPOLOGY statement
- When you add or change information in the plan, such as workstation definitions

To have the Symphony file rebuilt or renewed, you can use the Symphony Renew option of the Daily Planning menu (option 3.5 in previous Tivoli Workload Scheduler for z/OS ISPF panels).

This renew function can also be used to recover from error situations such as:

- A non-valid job definition in the script library
- Incorrect workstation definitions
- An incorrect Windows user name or password
- Changes to the script library or to the definitions of the TOPOLOGY statement

14.1.4 Making the end-to-end scheduling system fault tolerant

In the following, we cover some possible cases of failure in end-to-end scheduling and ways to mitigate against these failures:

1. The Tivoli Workload Scheduler for z/OS engine (controller) can fail due to a system or task outage.
2. The Tivoli Workload Scheduler for z/OS server can fail due to a system or task outage.
3. The domain managers at the first level (that is, the domain managers directly connected to the Tivoli Workload Scheduler for z/OS server), can fail due to a system or task outage.

To avoid an outage of the end-to-end workload managed in the Tivoli Workload Scheduler for z/OS engine and server and in the Tivoli Workload Scheduler domain manager, you should consider:

- Using a standby engine (controller) for the Tivoli Workload Scheduler for z/OS engine (controller).
- Making sure that your Tivoli Workload Scheduler for z/OS server can be reached if the Tivoli Workload Scheduler for z/OS engine (controller) is moved to one of its standby engines (TCP/IP configuration in your enterprise). *Remember that the end-to-end server started task always must be active on the same z/OS system as the active engine (controller).*
- Defining backup domain managers for your Tivoli Workload Scheduler domain managers at the first level.

> **Note:** It is a good practice to define backup domain managers for all domain managers in the Tivoli Workload Scheduler network.

Figure 14-10 on page 356 shows an example of a fault-tolerant end-to-end network with a Tivoli Workload Scheduler for z/OS standby controller engine and one Tivoli Workload Scheduler backup domain manager for one Tivoli Workload Scheduler domain manager at the first level.

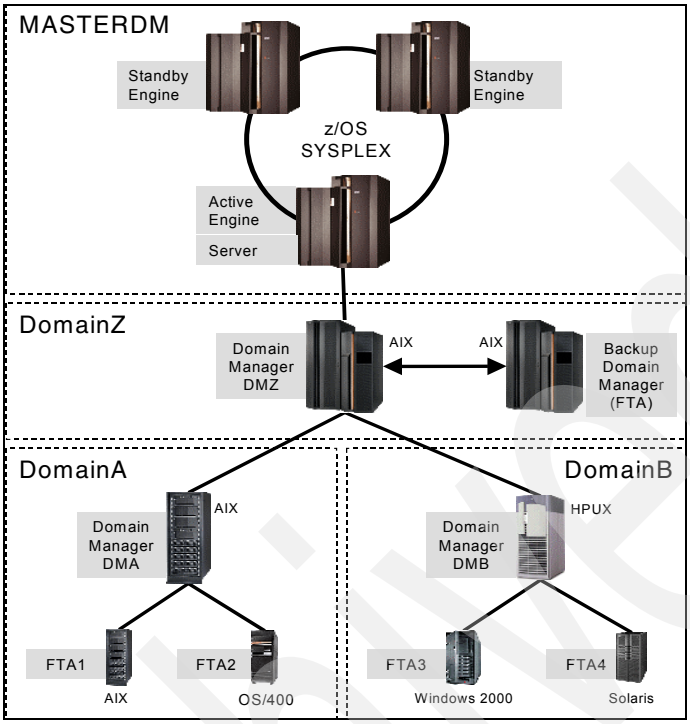


Figure 14-10 Redundant configuration with standby engine and Tivoli Workload Scheduler backup DM

If the domain manager for DomainZ fails, it will be possible to switch to the backup domain manager. The backup domain manager has an updated Symphony file and knows the subordinate domain managers and fault-tolerant agents, so it can take over the responsibilities of the domain manager. This switch can be performed without any outages in the workload management.

If the switch to the backup domain manager will be active across the Tivoli Workload Scheduler for z/OS plan extension, you must change the topology definitions in the Tivoli Workload Scheduler for z/OS DOMREC initialization statements. The backup domain manager fault-tolerant workstation will be the domain manager at the first level for the Tivoli Workload Scheduler network, even after the plan extension.

Example 14-1 on page 357 shows how to change the name of the fault-tolerant workstation in the DOMREC initialization statement, if the switch to the backup domain manager is effective across the Tivoli Workload Scheduler for z/OS plan extension.

Example 14-1 DOMREC initialization statement

```
DOMREC DOMAIN(DOMAINZ) DOMMGR(FDMZ) DOMPARENT(MASTERDM)

Should be changed to:

DOMREC DOMAIN(DOMAINZ) DOMMGR(FDMB) DOMPARENT(MASTERDM)

Where FDMB is the name of the fault tolerant workstation where the
backup domain manager is running.
```

If the Tivoli Workload Scheduler for z/OS engine or server fails, it will be possible to let one of the standby engines in the same sysplex take over. This takeover can be accomplished without any outages in the workload management.

The Tivoli Workload Scheduler for z/OS server must follow the Tivoli Workload Scheduler for z/OS engine. That is, if the Tivoli Workload Scheduler for z/OS engine is moved to another system in the sysplex, the Tivoli Workload Scheduler for z/OS server must be moved to the same system in the sysplex.

Note: The synchronization between the Symphony file on the Tivoli Workload Scheduler domain manager and the Symphony file on its backup domain manager has improved considerably with FixPack 04 for Tivoli Workload Scheduler, in which an enhanced and improved fault-tolerant switch manager functionality is introduced.

14.1.5 Benefits of end-to-end scheduling

The benefits that can be gained from using the Tivoli Workload Scheduler for z/OS end-to-end scheduling include:

- The ability to connect Tivoli Workload Scheduler fault-tolerant agents to a Tivoli Workload Scheduler for z/OS controller.
- Scheduling on additional operating systems.
- The ability to define resource dependencies between jobs that run on different FTAs or in different domains.
- Synchronizing work in mainframe and non-mainframe environments.
- The ability to organize the scheduling network into multiple tiers, delegating some responsibilities to Tivoli Workload Scheduler domain managers.

- Extended planning capabilities, such as the use of long-term plans, trial plans, and extended plans, also for the Tivoli Workload Scheduler network.

  "Extended plans" also means that the current plan can span more than 24 hours. One possible benefit is being able to extend a current plan over a time period when no one will be available to verify that the current plan was successfully created each day, such as over a holiday weekend. The end-to-end environment also allows extension of the current plan for a specified length of time, or replanning the current plan to remove completed jobs.
- Powerful run-cycle and calendar functions. Tivoli Workload Scheduler end-to-end enables more complex run cycles and rules to be defined to determine when a job stream should be scheduled.
- Ability to create a Trial Plan that can span more than 24 hours.
- Improved use of resources (keep resource if job ends in error).
- Enhanced use of host names instead of dotted IP addresses.
- Multiple job or job stream instances in the same plan. In the end-to-end environment, job streams are renamed using a unique identifier so that multiple job stream instances can be included in the current plan.
- The ability to use batch tools (for example, Batchloader, Massupdate, OCL, BCIT) that enable batched changes to be made to the Tivoli Workload Scheduler end-to-end database and plan.
- The ability to specify at the job level whether the job's script should be centralized (placed in Tivoli Workload Scheduler for z/OS JOBLIB) or non-centralized (placed locally on the Tivoli Workload Scheduler agent).
- Use of Tivoli Workload Scheduler for z/OS JCL variables in both centralized and non-centralized scripts.
- The ability to use Tivoli Workload Scheduler for z/OS recovery in centralized scripts or Tivoli Workload Scheduler recovery in non-centralized scripts.
- The ability to define and browse operator instructions associated with jobs in the database and plan. In a Tivoli Workload Scheduler environment, it is possible to insert comments or a description in a job definition, but these comments and description are not visible from the plan functions.
- The ability to define a job stream that will be submitted automatically to Tivoli Workload Scheduler when one of the following events occurs in the z/OS system: a particular job is executed or terminated in the z/OS system, a specified resource becomes available, or a z/OS data set is created or opened.

Considerations

Implementing Tivoli Workload Scheduler for z/OS end-to-end also imposes some limitations:

- Windows users' passwords are defined directly (without any encryption) in the Tivoli Workload Scheduler for z/OS server initialization parameters. It is possible to place these definitions in a separate library with restricted access (restricted by RACF, for example) to authorized persons.
- In an end-to-end scheduling network, some of the conman command options are disabled. On an end-to-end FTA, the conman command allows only display operations and the subset of commands (such as **kill**, **altpass**, **link**/**unlink**, **start**/**stop**, and **switchmgr**) that do not affect the status or sequence of jobs. Command options that could affect the information that is contained in the Symphony file are not allowed. For a complete list of allowed conman commands, refer to 14.5, "conman commands in the end-to-end environment" on page 377.
- Workstation classes are not supported in an end-to-end scheduling network.
- The LIMIT attribute is supported on the workstation level, not on the job stream level in an end-to-end environment.
- Some Tivoli Workload Scheduler functions are not available directly on Tivoli Workload Scheduler FTAs, but can be handled by other functions in Tivoli Workload Scheduler for z/OS.

  For example:

  - Tivoli Workload Scheduler prompts
    - Recovery prompts are supported.
    - The Tivoli Workload Scheduler predefined and ad hoc prompts can be replaced with the manual workstation function in Tivoli Workload Scheduler for z/OS.
  - Tivoli Workload Scheduler file dependencies
    - It is not possible to define file dependencies directly at job level in Tivoli Workload Scheduler for z/OS for FTA jobs.
    - The filewatch program that is delivered with Tivoli Workload Scheduler can be used to create file dependencies for FTA jobs in Tivoli Workload Scheduler for z/OS. A job runs the filewatch.sh script to check that the file dependency is "replaced" by a job dependency in which a predecessor job checks for the file using the filewatch program.
  - Dependencies on job stream level

    The traditional way to handle these types of dependencies in Tivoli Workload Scheduler for z/OS is to define a "dummy start" and "dummy end" job at the beginning and end of the job streams, respectively.

– Repeat range (that is, “rerun this job every 10 minutes”)

  Although there is no built-in function for this in Tivoli Workload Scheduler for z/OS, it can be accomplished in different ways, such as by defining the job repeatedly in the job stream with specific start times or by using a PIF (Tivoli Workload Scheduler for z/OS Programming Interface) program to rerun the job every 10 minutes.

– Job priority change

  Job priority cannot be changed directly for an individual fault-tolerant job. In an end-to-end configuration, it is possible to change the priority of a job stream. When the priority of a job stream is changed, all jobs within the job stream will have the same priority.

– Internetwork dependencies

  An end-to-end configuration supports dependencies on a job that is running in the same Tivoli Workload Scheduler end-to-end network.

14.2 Job Scheduling Console and related components

The Job Scheduling Console (JSC) provides another way of working with Tivoli Workload Scheduler for z/OS databases and current plan. The JSC is a graphical user interface that connects to the Tivoli Workload Scheduler for z/OS engine via a Tivoli Workload Scheduler for z/OS TCP/IP Server task. Usually this task is dedicated exclusively to handling JSC communications. Later in this book, the server task that is dedicated to JSC communications will be referred to as the *JSC Server* (Figure 14-11).

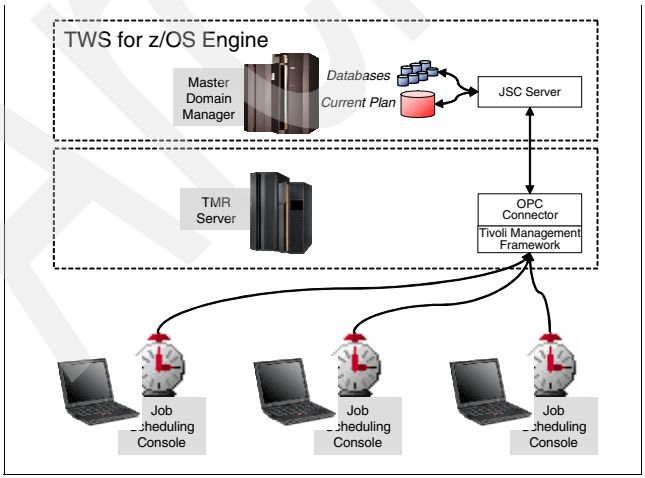


Figure 14-11 Communication via the JSC Server

The TCP/IP server is a separate address space, started and stopped automatically either by the engine or by the user via the z/OS start and stop commands. More than one TCP/IP server can be associated with an engine.

The Job Scheduling Console can be run on almost any platform. Using the JSC, an operator can access both Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS scheduling engines. In order to communicate with the scheduling engines, the JSC requires several additional components to be installed:

- Tivoli Management Framework
- Job Scheduling Services (JSS)
- Tivoli Workload Scheduler connector, Tivoli Workload Scheduler for z/OS connector, or both

The Job Scheduling Services and the connectors must be installed on top of the Tivoli Management Framework. Together, the Tivoli Management Framework, the Job Scheduling Services, and the connector provide the interface between JSC and the scheduling engine.

The Job Scheduling Console is installed locally on your desktop computer, laptop computer, or workstation.

14.2.1 A brief introduction to the Tivoli Management Framework

Tivoli Management Framework provides the foundation on which the Job Scheduling Services and connectors are installed. It also performs access verification when a Job Scheduling Console user logs in. The Tivoli Management Environment® (TME®) uses the concept of Tivoli Management Regions (TMRs). There is a single server for each TMR, called the TMR server; this is analogous to the Tivoli Workload Scheduler master server. The TMR server contains the Tivoli object repository (a database used by the TMR). Managed nodes are semi-independent agents that are installed on other nodes in the network; these are roughly analogous to Tivoli Workload Scheduler fault-tolerant agents. For more information about the Tivoli Management Framework, see the *IBM Tivoli Management Framework 4.1 User's Guide*, GC32-0805.

14.2.2 Job Scheduling Services (JSS)

The Job Scheduling Services component provides a unified interface in the Tivoli Management Framework for different job scheduling engines. Job Scheduling Services does not do anything on its own; it requires additional components called connectors in order to connect to job scheduling engines. It must be installed on either the TMR server or a managed node.

14.2.3 Connectors

Connectors are the components that enable the Job Scheduling Services to talk with different types of scheduling engines. When working with a particular type of scheduling engine, the Job Scheduling Console communicates with the scheduling engine via the Job Scheduling Services and the connector. A different connector is required for each type of scheduling engine. A connector can be installed only on a computer where the Tivoli Management Framework and Job Scheduling Services have already been installed.

There are two types of connectors for connecting to the two types of scheduling engines in the Tivoli Workload Scheduler 8.2 suite:

- Tivoli Workload Scheduler for z/OS connector (or OPC connector)
- Tivoli Workload Scheduler Connector

Job Scheduling Services communicates with the engine via the Connector of the appropriate type. When working with a Tivoli Workload Scheduler for z/OS engine, the JSC communicates via the Tivoli Workload Scheduler for z/OS Connector. When working with a Tivoli Workload Scheduler engine, the JSC communicates via the Tivoli Workload Scheduler Connector.

The two types of connectors function somewhat differently: The Tivoli Workload Scheduler for z/OS Connector communicates over TCP/IP with the Tivoli Workload Scheduler for z/OS engine running on a mainframe (MVS or z/OS) computer. The Tivoli Workload Scheduler Connector performs direct reads and writes of the Tivoli Workload Scheduler plan and database files on the same computer as where the Tivoli Workload Scheduler Connector runs.

A Connector instance must be created before the Connector can be used. Each type of Connector can have multiple instances. A separate instance is required for each engine that will be controlled by JSC.

We now discuss each type of Connector in more detail. See Figure 14-12 on page 364.

Tivoli Workload Scheduler for z/OS Connector

Also sometimes called the OPC Connector, the Tivoli Workload Scheduler for z/OS Connector can be instantiated on any TMR server or managed node. The Tivoli Workload Scheduler for z/OS Connector instance communicates via TCP with the Tivoli Workload Scheduler for z/OS TCP/IP server. You might, for example, have two different Tivoli Workload Scheduler for z/OS engines that both must be accessible from the Job Scheduling Console. In this case, you would install one Connector instance for working with one Tivoli Workload Scheduler for z/OS engine, and another Connector instance for communicating with the other engine. When a Tivoli Workload Scheduler for z/OS Connector instance is

created, the IP address (or host name) and TCP port number of the Tivoli Workload Scheduler for z/OS engine's TCP/IP server are specified. The Tivoli Workload Scheduler for z/OS Connector uses these two pieces of information to connect to the Tivoli Workload Scheduler for z/OS engine.

Tivoli Workload Scheduler Connector

The Tivoli Workload Scheduler Connector must be instantiated on the host where the Tivoli Workload Scheduler engine is installed so that it can access the plan and database files locally. This means that the Tivoli Management Framework must be installed (either as a TMR server or managed node) on the server where the Tivoli Workload Scheduler engine resides. Usually, this server is the Tivoli Workload Scheduler master domain manager. But it may also be desirable to connect with JSC to another domain manager or to a fault-tolerant agent. If multiple instances of Tivoli Workload Scheduler are installed on a server, it is possible to have one Tivoli Workload Scheduler Connector instance for each Tivoli Workload Scheduler instance on the server. When a Tivoli Workload Scheduler Connector instance is created, the full path to the Tivoli Workload Scheduler home directory associated with that Tivoli Workload Scheduler instance is specified. This is how the Tivoli Workload Scheduler Connector knows where to find the Tivoli Workload Scheduler databases and plan.

Connector instances

The following examples show how Connector instances might be installed in the real world.

One Connector instance of each type

In Figure 14-12, there are two Connector instances, including one Tivoli Workload Scheduler for z/OS Connector instance and one Tivoli Workload Scheduler Connector instance:

- The Tivoli Workload Scheduler for z/OS Connector instance is associated with a Tivoli Workload Scheduler for z/OS engine running in a remote sysplex. Communication between the Connector instance and the remote scheduling engine is conducted over a TCP connection.
- The Tivoli Workload Scheduler Connector instance is associated with a Tivoli Workload Scheduler engine installed on the same AIX server. The Tivoli Workload Scheduler Connector instance reads from and writes to the plan (the Symphony file) of the Tivoli Workload Scheduler engine.

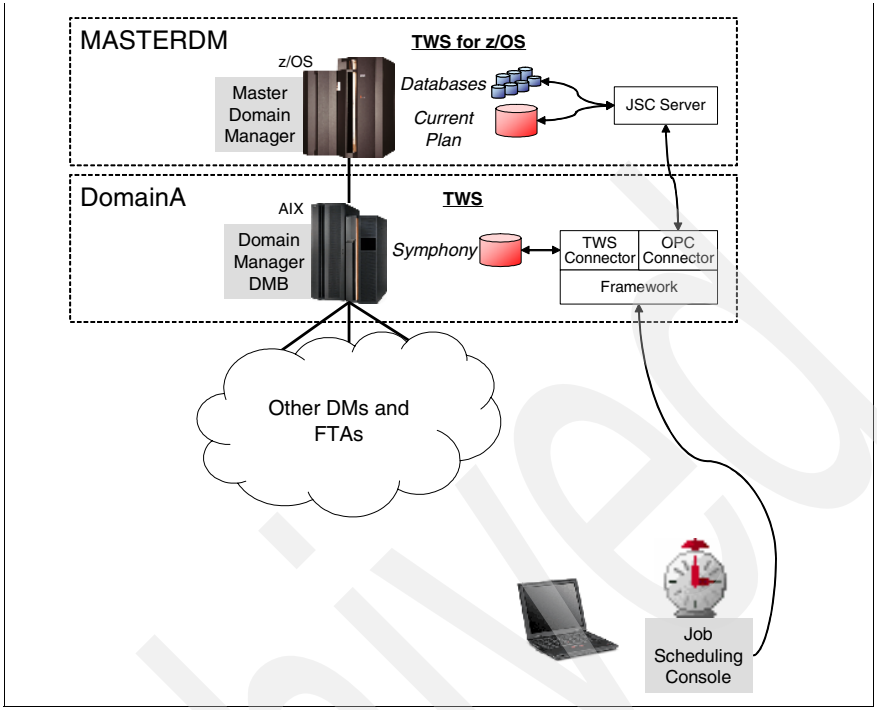


Figure 14-12 One Tivoli Workload Scheduler for z/OS Connector and one Tivoli Workload Scheduler Connector instance

Tip: Tivoli Workload Scheduler Connector instances must be created on the server where the Tivoli Workload Scheduler engine is installed because the Connector must be able to have access locally to the Tivoli Workload Scheduler engine (specifically, to the plan and database files). This limitation obviously does not apply to Tivoli Workload Scheduler for z/OS Connector instances because the Tivoli Workload Scheduler for z/OS Connector communicates with the remote Tivoli Workload Scheduler for z/OS engine over TCP/IP.

In this example, the Connectors are installed on the domain manager DMB. This domain manager has one Connector instance of each type:

- A Tivoli Workload Scheduler Connector to monitor the plan file (Symphony) locally on DMB

- A Tivoli Workload Scheduler for z/OS (OPC) Connector to work with the databases and current plan on the mainframe

Having the Tivoli Workload Scheduler Connector installed on a DM provides the operator with the ability to use JSC to look directly at the Symphony file on that workstation. This is particularly useful in the event that problems arise during the production day. If any discrepancy appears between the state of a job in the Tivoli Workload Scheduler for z/OS current plan and the Symphony file on an FTA, it is useful to be able to look at the Symphony file directly. Another benefit is that retrieval of job logs from an FTA is much faster when the job log is retrieved through the Tivoli Workload Scheduler Connector. If the job log is fetched through the Tivoli Workload Scheduler for z/OS engine, it can take much longer.

Connectors on multiple domain managers

With the previous version of Tivoli Workload Scheduler (Version 8.1) it was necessary to have a single primary domain manager that was the parent of all other domain managers. Figure 14-12 on page 364 shows an example of such an arrangement. Tivoli Workload Scheduler 8.2 removes this limitation. With Version 8.2, it is possible to have more than one domain manager directly under the master domain manager. Most end-to-end scheduling networks will have more than one domain manager under the master. For this reason, it is a good idea to install the Tivoli Workload Scheduler Connector and OPC Connector on more than one domain manager.

Note: It is a good idea to set up more than one Tivoli Workload Scheduler for z/OS Connector instance associated with the engine (as in Figure 14-13). This way, if there is a problem with one of the workstations running the Connector, JSC users will still be able to access the Tivoli Workload Scheduler for z/OS engine via the other Connector. If JSC access is important to your enterprise, it is vital to set up redundant Connector instances like this.

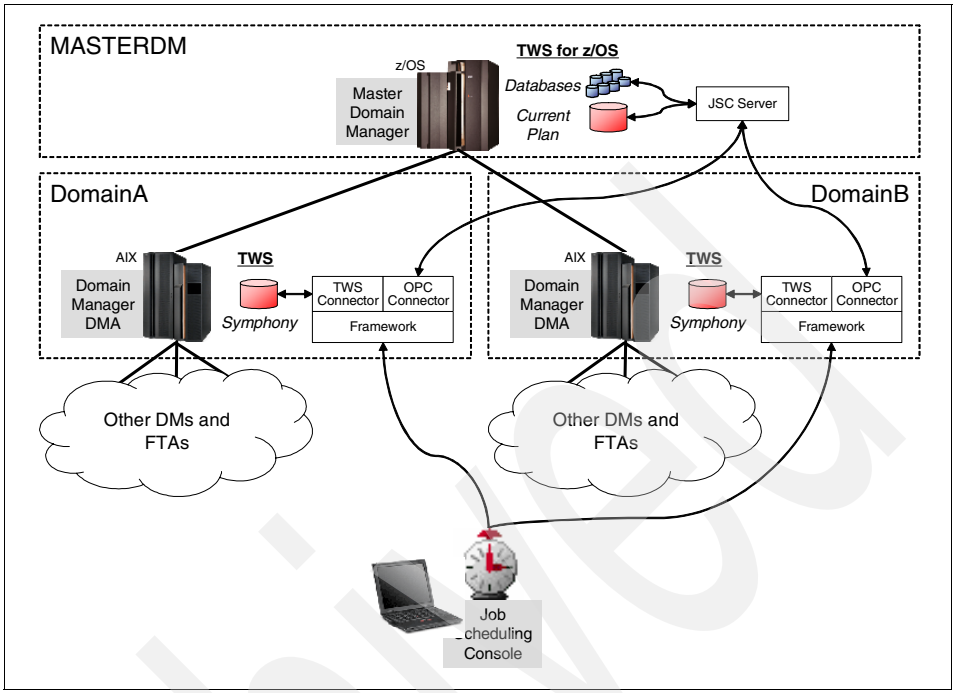


Figure 14-13 An example with two Connector instances of each type

Next, we discuss the Connectors in more detail.

The Connector programs

These are the programs that run *behind the scenes* to make the Connectors work. We describe each program and its function.

Programs of the Tivoli Workload Scheduler for z/OS Connector

The programs that comprise the Tivoli Workload Scheduler for z/OS Connector are located in $BINDIR/OPC (Figure 14-14 on page 367).

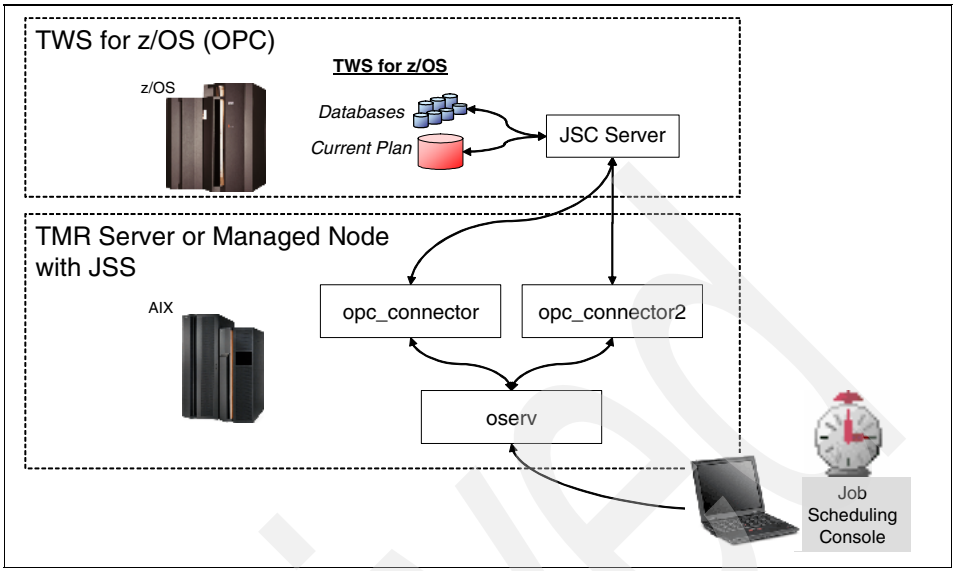


Figure 14-14 Programs of the Tivoli Workload Scheduler for z/OS (OPC) Connector

- opc_connector

  The main Connector program that contains the implementation of the main Connector methods (basically all methods that are required to connect to and retrieve data from Tivoli Workload Scheduler for z/OS engine). It is implemented as a threaded daemon, which means that it is automatically started by the Tivoli Framework at the first request that should be handled by it, and it will stay active until there has not been a request for a long time. After it is started, it handles starting new threads for all JSC requests that require data from a specific Tivoli Workload Scheduler for z/OS engine.

- opc_connector2

  A small Connector program that contains the implementation for small methods that do not require data from Tivoli Workload Scheduler for z/OS. This program is implemented *per method*, which means that Tivoli Framework starts this program when a method implemented by it is called, the process performs the action for this method, and then is terminated. This is useful for methods (like the ones called by JSC when it starts and asks for information from all of the Connectors) that can be isolated and not logical to maintain the process activity.

Programs of the Tivoli Workload Scheduler Connector

The programs that comprise the Tivoli Workload Scheduler Connector are located in $BINDIR/Maestro (Figure 14-15).

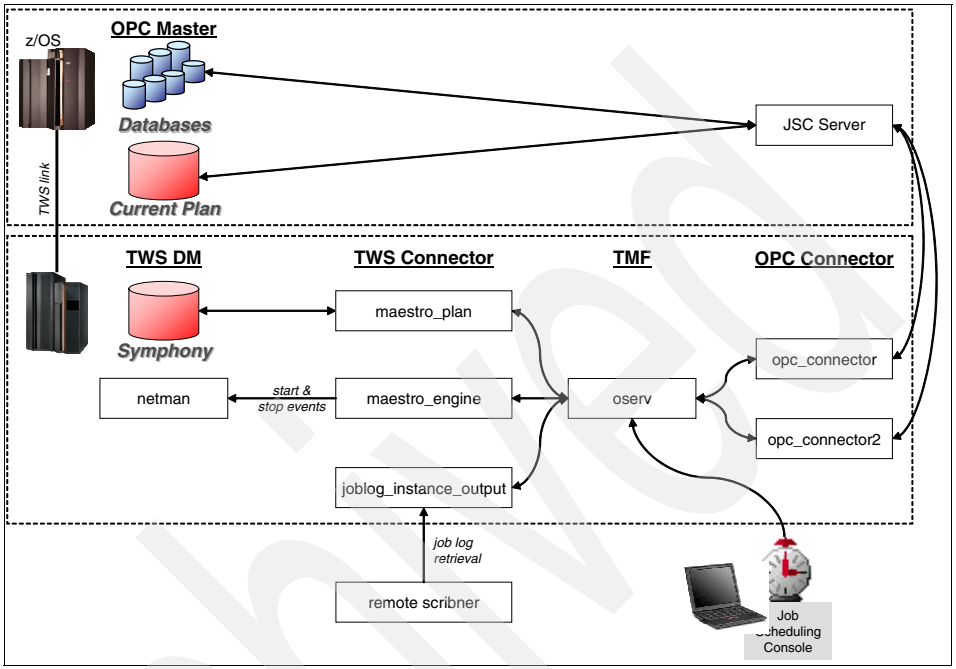


Figure 14-15 Programs of the Tivoli Workload Scheduler Connector and the Tivoli Workload Scheduler for z/OS Connector

- maestro_engine

  The maestro_engine program performs authentication when a user logs on via the Job Scheduling Console. It also starts and stops the Tivoli Workload Scheduler engine. It is started by the Tivoli Management Framework (specifically, the oserv program) when a user logs on from JSC. It terminates after 30 minutes of inactivity.

  Note: oserv is the Tivoli service that is used as the object request broker (ORB). This service runs on the Tivoli management region server and each managed node.

- maestro_plan

  The maestro_plan program reads from and writes to the Tivoli Workload Scheduler plan. It also handles switching to a different plan. The program is

started when a user accesses the plan. It terminates after 30 minutes of inactivity.

Note: The maestro_database program is used only on Tivoli Workload Scheduler master domain managers. In an end-to-end scheduling network, the Tivoli Workload Scheduler for z/OS controller and server act as the MDM for the whole scheduling network, so the maestro_database program is not used.

- job_instance_output

  The job_instance_output program retrieves job standard list files. It is started when a JSC user runs the Browse Job Log operation. It starts up, retrieves the requested stdlist file, and then terminates.

14.3 Job log retrieval in an end-to-end environment

In this section, we cover the detailed steps of job log retrieval in an end-to-end environment using the JSC. The steps differ, depending on which Connector you are using to retrieve the job log and whether the firewalls are involved. We cover all of these scenarios: using the Tivoli Workload Scheduler Connector on a domain manager or, using the Tivoli Workload Scheduler for z/OS (OPC) Connector, and with the firewalls in the picture.

14.3.1 Job log retrieval via the Tivoli Workload Scheduler Connector

As shown in Figure 14-16 on page 370, the steps behind the scenes in an end-to-end scheduling network when retrieving the job log via the domain manager (using the Tivoli Workload Scheduler Connector) are:

1. Operator requests joblog in Job Scheduling Console.
2. JSC connects to oserv running on the domain manager.
3. oserv spawns job_instance_output to fetch the job log.
4. job_instance_output communicates over TCP directly with the workstation where the joblog exists, bypassing the domain manager.
5. netman on that workstation spawns scribner and hands over the TCP connection with job_instance_output to the new scribner process.
6. scribner retrieves the joblog.
7. scribner sends the joblog to job_instance_output on the master.
8. job_instance_ouput relays the job log to oserv.

9. oserv sends the job log to JSC.

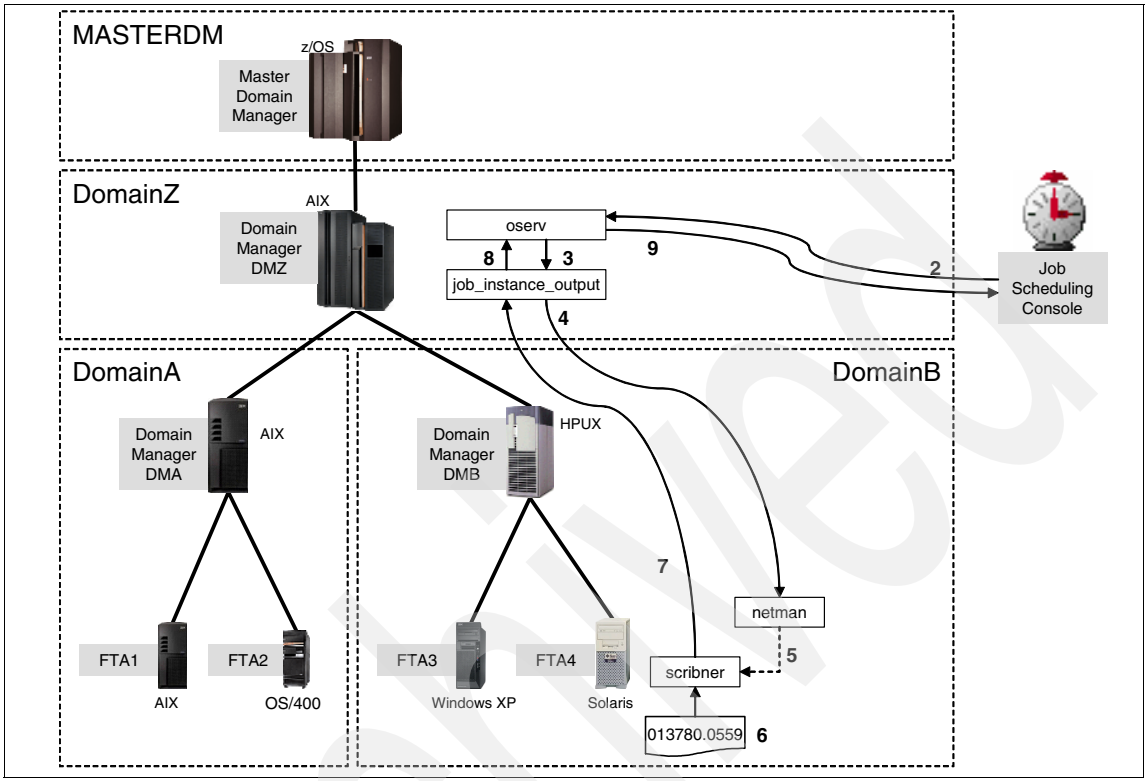


Figure 14-16 Job log retrieval in an end-to-end scheduling network via the domain manager

14.3.2 Job log retrieval via the OPC Connector

As shown in Figure 14-17 on page 372, the following steps take place behind the scenes in an end-to-end scheduling network when retrieving the job log using the OPC Connector.

The initial request for joblog is done:

1. Operator requests joblog in Job Scheduling Console.
2. JSC connects to oserv running on the domain manager.
3. oserv tells the OPC Connector program to request the joblog from the OPC system.
4. opc_connector relays the request to the JSC Server task on the mainframe.
5. The JSC Server requests the job log from the controller.

The next step depends on whether the job log has already been retrieved. If so, skip to step 17. If the job log has not been retrieved yet, continue with step 6.

Assuming that the log has not been retrieved already:

6. The controller sends the request for the joblog to the sender subtask.
7. The controller sends a message to the operator indicating that the job log has been requested. This message is displayed in a dialog box in JSC. (The message is sent via this path: Controller → JSC Server → opc_connector → oserv → JSC).
8. The sender subtask sends the request to the output translator, via the output queue.
9. The output translator thread reads the request and spawns a job log retriever thread to handle it.
10. The job log retriever thread opens a TCP connection directly to the workstation where the job log exists, bypassing the domain manager.
11. netman on that workstation spawns scribner and hands over the TCP connection with the job log retriever to the new scribner process.
12. scribner retrieves the job log.
13. scribner sends the joblog to the job log retriever thread.
14. The job log retriever thread passes the job log to the input writer thread.
15. The input writer thread sends the job log to the receiver subtask, via the input queue.
16. The receiver subtask sends the job log to the controller.

When the operator requests the job log a second time, the first five steps are the same as in the initial request (above). This time around, because the job log has already been received by the controller:

17. The controller sends the job log to the JSC Server.
18. The JSC Server sends the information to the OPC connector program running on the domain manager.
19. The Tivoli Workload Scheduler for z/OS Connector relays the job log to oserv.
20. oserv relays the job log to JSC and JSC displays the job log in a new window.

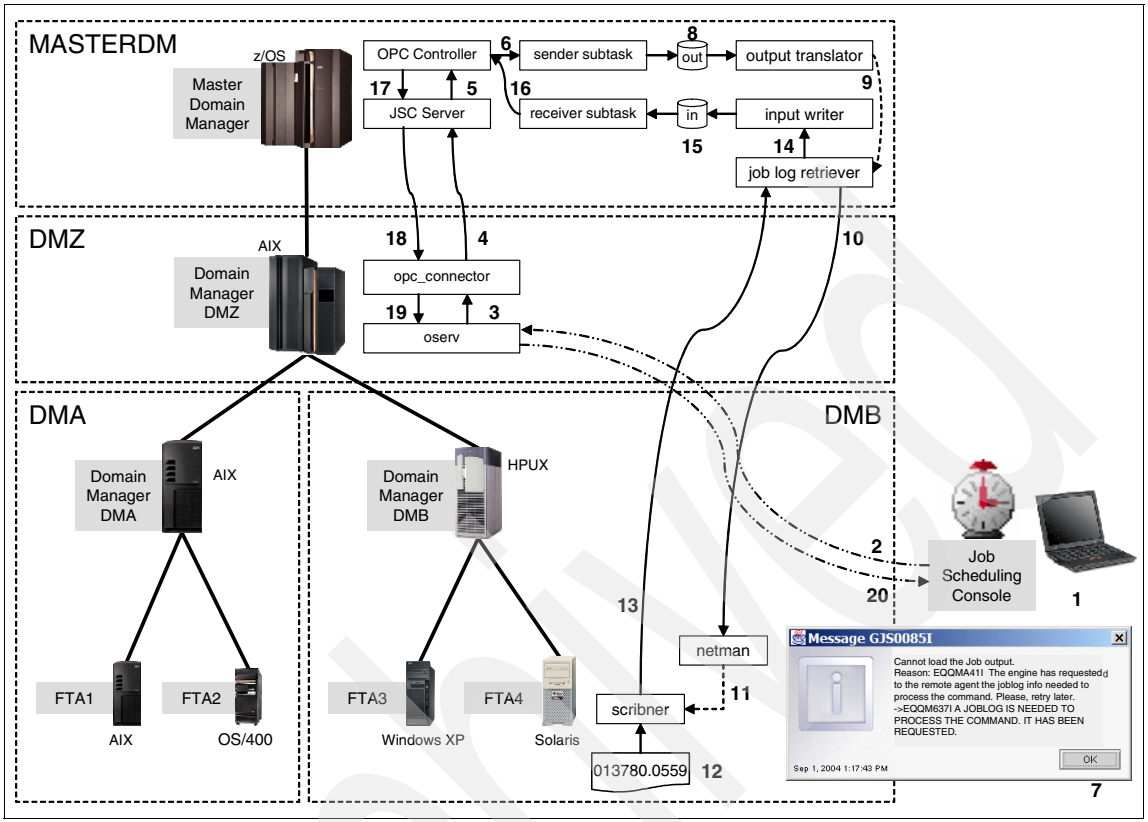


Figure 14-17 Job log retrieval in an end-to-end network via the Tivoli Workload Scheduler for z/OS- no FIREWALL=Y configured

14.3.3 Job log retrieval when firewalls are involved

When the firewalls are involved (that is, FIREWALL=Y configured in the CPUREC definition of the workstation in which the job log is retrieved), the steps for retrieving the job log in an end-to-end scheduling network are different. These steps are shown in Figure 14-18 on page 373. Note that the firewall is configured to allow only the following traffic: DMY → DMA and DMZ → DMB.

1. The operator requests the job log in JSC or the mainframe ISPF panels.
2. TCP connection is opened to the parent domain manager of the workstation where the job log exists.
3. netman on that workstation spawns router and hands over the TCP socket to the new router process.

4. router opens a TCP connection to netman on the parent domain manager of the workstation where the job log exists, because this DM is also behind the firewall.
5. netman on the DM spawns router and hands over the TCP socket with router to the new router process.
6. router opens a TCP connection to netman on the workstation where the job log exists.
7. netman on that workstation spawns scribner and hands over the TCP socket with router to the new scribner process.
8. scribner retrieves the job log.
9. scribner on FTA4 sends the job log to router on DMB.
10. router sends the job log to the router program running on DMZ.

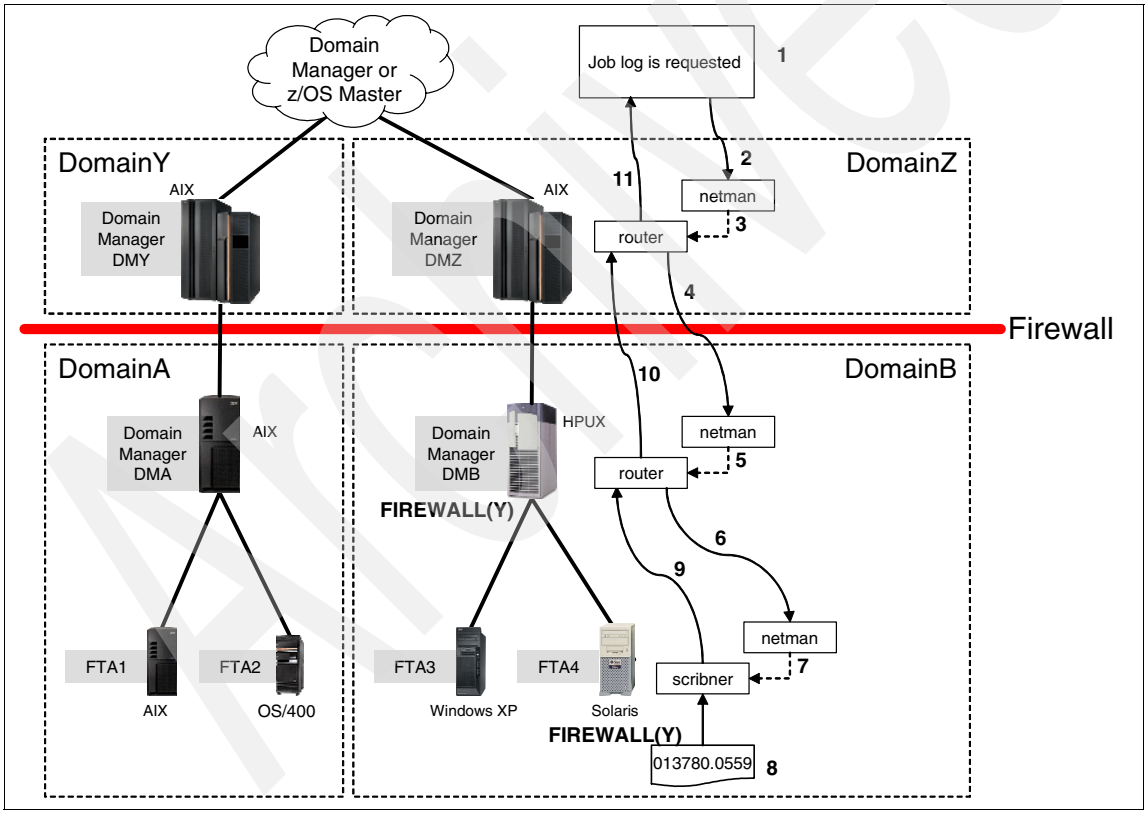


Figure 14-18 Job log retrieval with FIREWALL=Y configured

It is important to note that in the previous scenario, you should not configure the domain manager DMB as FIREWALL=N in its CPUREC definition. If you do, you

will not be able to retrieve the job log from FTA4, even though FTA4 is configured as FIREWALL=Y. This is shown is Figure 14-19.

In this case, when the TCP connection to the parent domain manager of the workstation where the job log exists (DMB) is blocked by the firewall, the connection request is not received by netman on DMB. The firewall does not allow direct connections from DMZ to FTA4. The only connections from DMZ that are permitted are those that go to DMB. Because DMB has FIREWALL=N, the connection did not go through DMZ; it tried to go straight to FTA4.

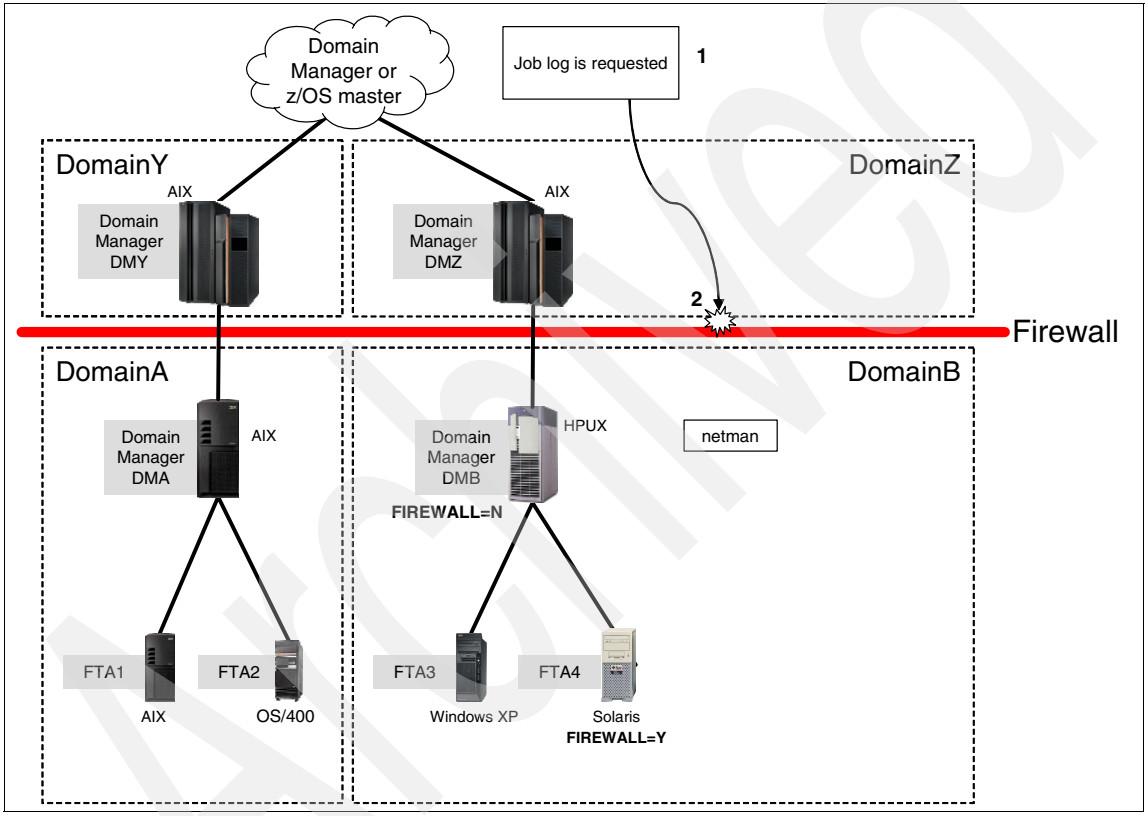


Figure 14-19 Wrong configuration: connection blocked

14.4 Tivoli Workload Scheduler, important files, and directory structure

Figure 14-20 shows the most important files in the Tivoli Workload Scheduler 8.2 working directory in USS (WRKDIR).

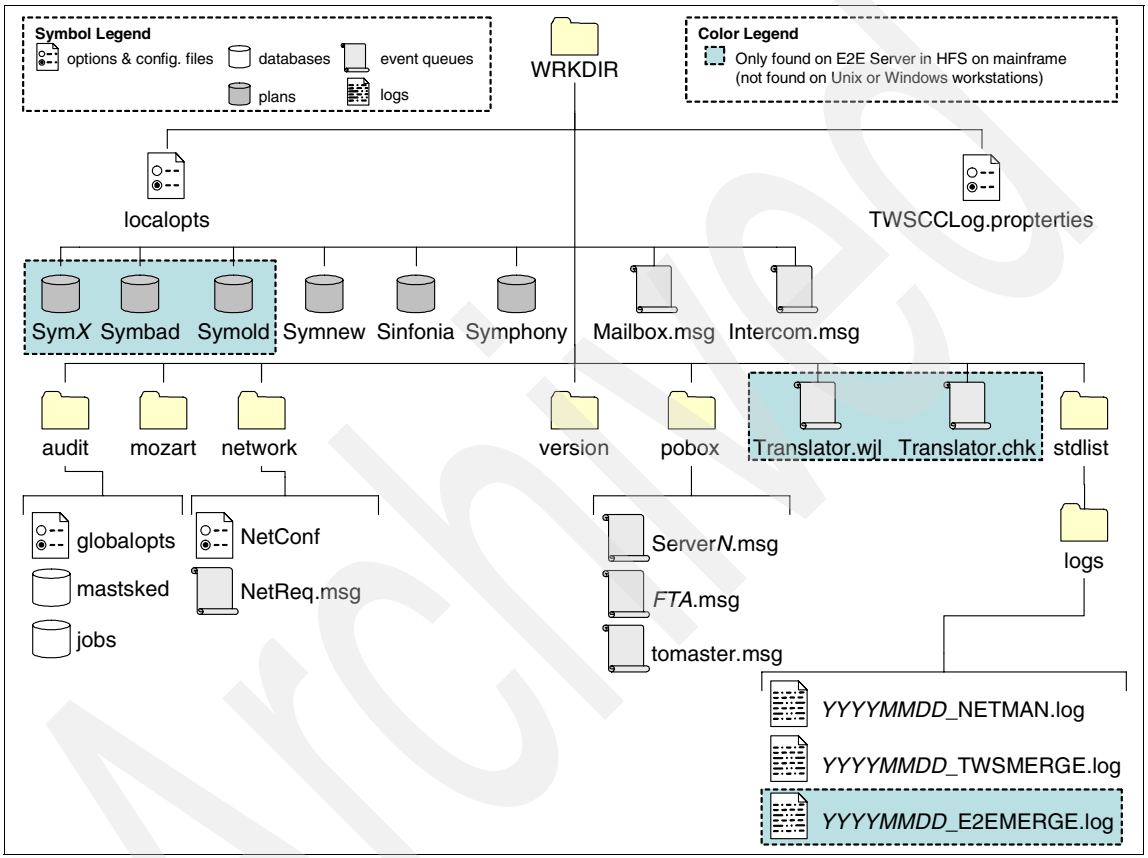


Figure 14-20 The most important files in the Tivoli Workload Scheduler 8.2 working directory in USS

The descriptions of the files are:

Sym*X* (where *X* is the name of the user who ran the CP extend or Symphony renew job): A temporary file created during a CP extend or Symphony renew. This file is copied to Symnew, which is then copied to Sinfonia and Symphony.

Symbad (Bad Symphony) Only created if CP extend or Symphony renew results in an invalid Symphony.

Symold (Old Symphony)	From prior to most recent CP extend or Symphony renew.
Translator.wjl	Translator event log for requested job logs.
Translator.chk	Translator checkpoint file.
YYYYMMDD_E2EMERGE.log	Translator log.

Note: The Symnew, SymX, and Symbad files are temporary files and normally cannot be seen in the USS work directory.

Figure 14-21 on page 376 shows the most important files in the Tivoli Workload Scheduler 8.2 binary directory in USS (BINDIR). The options files in the config subdirectory are only reference copies of these files; they are not active configuration files.

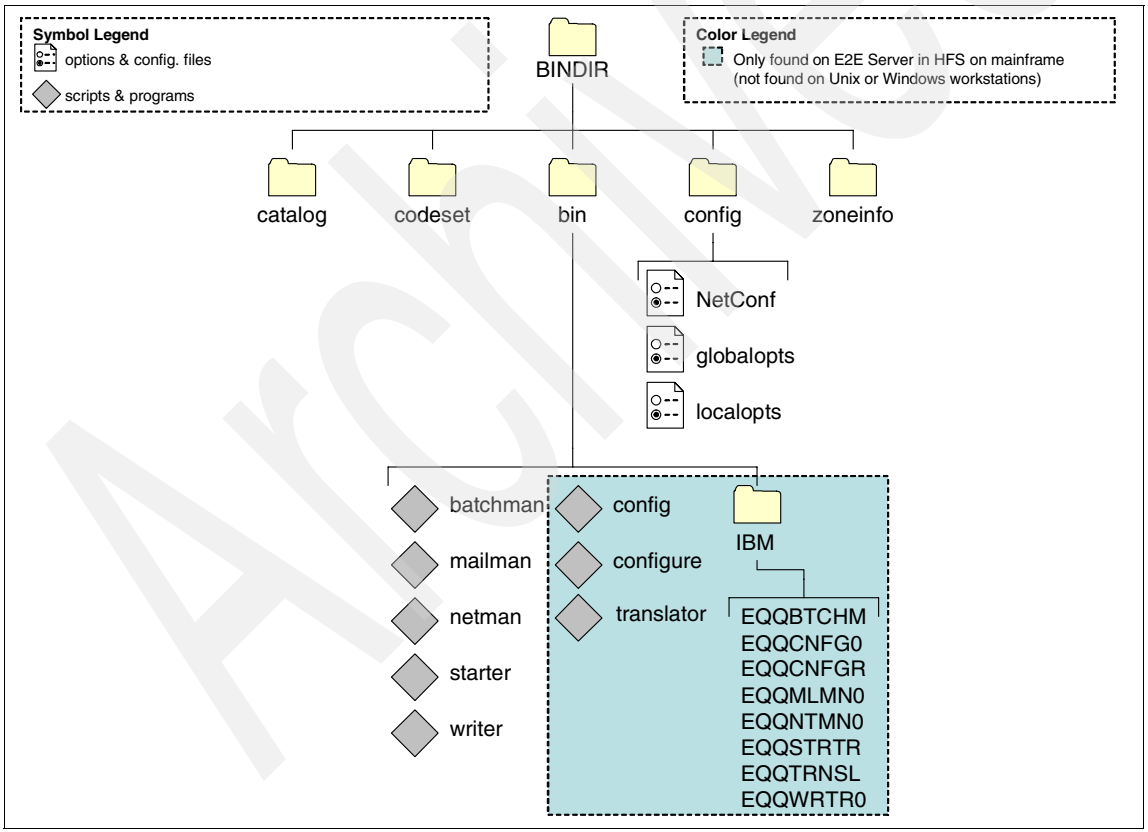


Figure 14-21 A list of the most important files in the Tivoli Workload Scheduler 8.2 binary directory in USS

Figure 14-22 shows the Tivoli Workload Scheduler 8.2 directory structure on the fault-tolerant agents. Note that the database files (such as jobs and calendars) are not used in the Tivoli Workload Scheduler 8.2 end-to-end scheduling environment.

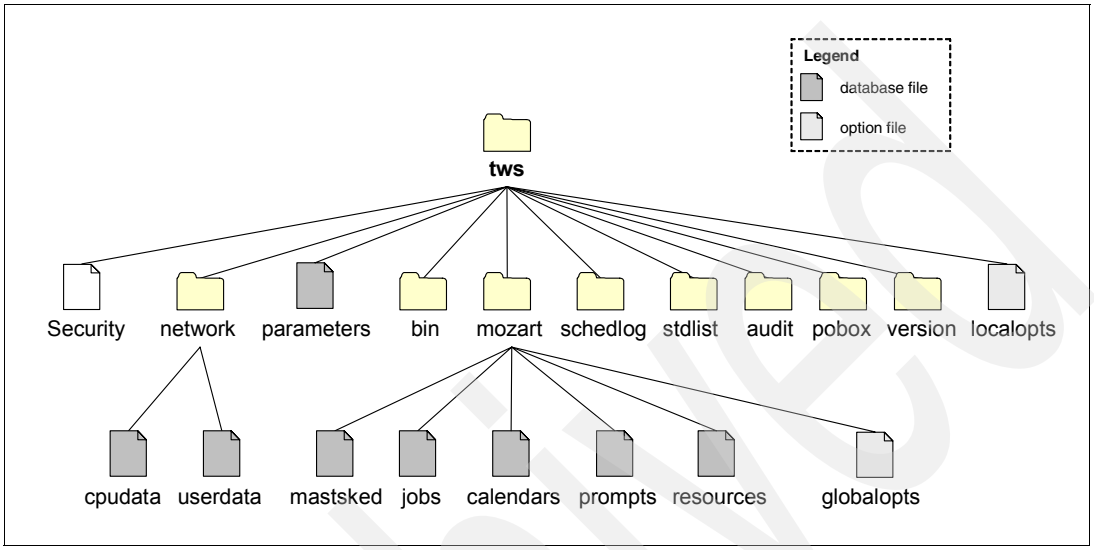


Figure 14-22 Tivoli Workload Scheduler 8.2 directory structure on the fault-tolerant agents

14.5 conman commands in the end-to-end environment

In Tivoli Workload Scheduler, you can use the conman command line interface to manage the production. A subset of these commands can also be used in an end-to-end scheduling network. In general, command options that could change the information contained in the Symphony file are not allowed. Disallowed conman command options include the addition or removal of dependencies, and the submission or cancellation of jobs.

Figure 14-23 on page 378 lists the conman commands that are available on end-to-end fault-tolerant workstations in a Tivoli Workload Scheduler 8.2 end-to-end scheduling network. Note that in the Type field, M stands for domain managers, F for fault-tolerant agents, and A for standard agents.

Note: The composer command line interface, used to manage database objects in a Tivoli Workload Scheduler environment, is not used in an end-to-end scheduling network. This is because the databases are located on the Tivoli Workload Scheduler for z/OS master.

Command	Description	Type
altpass	Alters a User object definition password.	D,F
console	Assigns the Workload Scheduler console.	D,F,S
continue	Ignores the next error.	D,F,S
display	Displays job streams.	D,F S
exit	Terminates Conman.	D,F,S
fence	Sets Workload Scheduler's job fence.	D,F,S
help	Displays command information.	D,F,S
kill	Stops an executing job.	D,F
limit	Changes a workstation limit.	D,F,S
link	Opens workstation links.	D,F,S
listsym	Displays a list of Symphony log files.	D,F
redo	Edits the previous command.	D,F,S
reply	Replies to a recovery prompt	D,F,S
setsym	Selects a Symphony log file.	D,F
showcpus	Displays workstation and link information.	D,F,S
showdomain	Displays domain information.	D,F,S
showdomain	Displays domain information.	D,F,S
showfiles	Displays information about files.	D,F
showjobs	Displays information about jobs.	D,F
showprompts	Displays information about prompts.	D,F
showresources	Displays information about resources.	D,F
showschedules	Displays information about job streams.	D,F
shutdown	Stops Workload Scheduler's production processes.	D,F,S
start	Starts Workload Scheduler's production processes.	D,F,S
status	Displays Workload Scheduler's production status.	D,F,S
stop	Stops Workload Scheduler's production processes.	D,F,S
switchmgr	Switches the domain manager.	D,F
sys-command	Sends a command to the system.	D,F,S
tellop	Sends a message to the console.	D,F,S
unlink	Closes workstation links.	D,F,S
version	Displays Conman's program banner.	D,F,S

Workstation types
D domain managers
F fault tolerant agents
S standard agents

Figure 14-23 conman commands available in end-to-end environment

15

TWS for z/OS end-to-end scheduling installation and customization

In this chapter, the following topics are discussed for the installation of Tivoli Workload Scheduler V8.2 in an end-to-end environment:

- Installing Tivoli Workload Scheduler for z/OS end-to-end scheduling
- Installing FTAs in an end-to-end environment
- Define, activate, verify fault-tolerant workstations
- Creating fault-tolerant workstation job definitions and job streams
- Verification test of end-to-end scheduling

15.1 Installing Tivoli Workload Scheduler for z/OS end-to-end scheduling

In this section, we guide you though the installation process of Tivoli Workload Scheduler for z/OS end-to-end feature. (The installation process for Tivoli Workload Scheduler for z/OS is discussed in Chapter 1, “Tivoli Workload Scheduler for z/OS installation” on page 3.)

To activate support for end-to-end scheduling in Tivoli Workload Scheduler for z/OS to be able to schedule jobs on the Tivoli Workload Scheduler FTAs, follow these steps:

1. Run EQQJOBS and specify `Y` for the end-to-end feature.

   See 15.1.1, “Executing EQQJOBS installation aid” on page 382.

2. Define controller (engine) and tracker (agent) subsystems in SYS1.PARMLIB.

   See 15.1.2, “Defining Tivoli Workload Scheduler for z/OS subsystems” on page 387.

3. Allocate the end-to-end data sets running the EQQPCS06 sample generated by EQQJOBS.

   See 15.1.3, “Allocate end-to-end data sets” on page 388.

4. Create and customize the work directory by running the EQQPCS05 sample generated by EQQJOBS.

   See 15.1.4, “Create and customize the work directory” on page 390.

5. Create started task procedures for Tivoli Workload Scheduler for z/OS.

   See 15.1.5, “Create started task procedures” on page 393.

6. Define workstation (CPU) configuration and domain organization by using the CPUREC and DOMREC statements in a new PARMLIB member. (The default member name is TPLGINFO.)

   See 15.1.6, “Initialization statements for Tivoli Workload Scheduler for z/OS end-to-end scheduling” on page 394, “DOMREC statement” on page 404, “CPUREC statement” on page 406, and Figure 15-6 on page 396.

7. Define Windows user IDs and passwords by using the USRREC statement in a new PARMLIB member. (The default member name is USRINFO.)

   It is important to remember that you have to define Windows user IDs and passwords only if you have fault-tolerant agents on Windows-supported platforms and want to schedule jobs to be run on these Windows platforms.

   See “USRREC statement” on page 413.

8. Define the end-to-end configuration by using the TOPOLOGY statement in a new PARMLIB member. (The default member name is TPLGPARM.) See “TOPOLOGY statement” on page 398.

   In the TOPOLOGY statement, you should make these specifications:

   – For the TPLGYMEM keyword, write the name of the member used in step 6 on page 380. (See Figure 15-6 on page 396.)
   – For the USRMEM keyword, write the name of the member used in step 7 on page 380. (See Figure 15-6 on page 396.)

9. Add the TPLGYSRV keyword to the OPCOPTS statement in the Tivoli Workload Scheduler for z/OS controller to specify the server name that will be used for end-to-end communication.

   See “OPCOPTS TPLGYSRV(server_name)” on page 396.

10. Add the TPLGYPRM keyword to the SERVOPTS statement in the Tivoli Workload Scheduler for z/OS end-to-end server to specify the member name used in step 8. This step activates end-to-end communication in the end-to-end server started task.

    See “SERVOPTS TPLGYPRM(member name/TPLGPARM)” on page 397.

11. Add the TPLGYPRM keyword to the BATCHOPT statement to specify the member name used in step 8. This step activates the end-to-end feature in the plan extend, plan replan, and Symphony renew batch jobs.

    See “TPLGYPRM(member name/TPLGPARM) in BATCHOPT” on page 397.

12. Optionally, you can customize the way the job name is generated in the Symphony file by the Tivoli Workload Scheduler for z/OS plan extend, replan, and Symphony renew batch jobs.

    The job name in the Symphony file can be tailored or customized by the JTOPTS TWSJOBNAME() parameter. See 15.1.9, “The JTOPTS TWSJOBNAME() parameter” on page 418 for more information.

    If you decide to customize the job name layout in the Symphony file, be aware that it can require that you reallocate the EQQTWSOU data set with larger record length. See “End-to-end input and output data sets” on page 388 for more information.

    Note: The JTOPTS TWSJOBNAME() parameter was introduced by APAR PQ77970.

13. Verify that the Tivoli Workload Scheduler for z/OS controller and server started tasks can be started (or restarted if already running) and verify that everything comes up correctly.

    Verification is described in 15.1.10, “Verify end-to-end installation” on page 422.

15.1.1 Executing EQQJOBS installation aid

EQQJOBS is a CLIST-driven ISPF dialog that can help you install Tivoli Workload Scheduler for z/OS. EQQJOBS assists in the installation of the engine and agent by building batch-job JCL that is tailored to your requirements. To make EQQJOBS executable, allocate these libraries to the DD statements in your TSO session:

- SEQQCLIB to SYSPROC
- SEQQPNL0 to ISPPLIB
- SEQQSKL0 and SEQQSAMP to ISPSLIB

Use EQQJOBS installation aid as follows:

1. To invoke EQQJOBS, enter the TSO command EQQJOBS from an ISPF environment. The primary panel shown in Figure 15-1 appears.

```
EQQJOBS0 ------------ EQQJOBS application menu  --------------
Select option ===>


   1  - Create sample job JCL


   2  - Generate OPC batch-job skeletons


   3  - Generate OPC Data Store samples


   X  - Exit from the EQQJOBS dialog
```

Figure 15-1 EQQJOBS primary panel

You only need to select options 1 and 2 for end-to-end specifications. We do not want to step through the whole EQQJOBS dialog so, instead, we show only the related end-to-end panels. (The referenced panel names are indicated in the top-left corner of the screens, as shown in Figure 15-1.)

2. Select option 1 in panel EQQJOBS0 (and press Enter twice), and make your necessary input into panel ID EQQJOBS8 (Figure 15-2).

```
EQQJOBS8---------------------------- Create sample job JCL --------------------
Command ===>

  END TO END FEATURE:                Y        (Y= Yes ,N= No)
   Installation Directory       ===> /usr/lpp/TWS/V8R2M0_____________________
                                ===> _____________________________________________
                                ===> _____________________________________________
   Work Directory               ===> /var/inst/TWS___________________________
                                ===> _____________________________________________
                                ===> _____________________________________________
   User for OPC address space   ===> UID  ___
   Refresh CP group             ===> GID   __

  RESTART AND CLEANUP (DATA STORE)  N        (Y= Yes ,N= No)
   Reserved destination         ===> OPC_____
   Connection type              ===> SNA      (SNA/XCF)
   SNA Data Store luname        ===> ________ (only for SNA connection  )
   SNA FN task luname           ===> ________ (only for SNA connection  )
   Xcf Group                    ===> ________ (only for XCF connection  )
   Xcf Data store member        ===> ________ (only for XCF connection  )
   Xcf FL task member           ===> ________ (only for XCF connection  )

Press ENTER to create sample job JCL
```

Figure 15-2 Server-related input panel

The following definitions are important:

- End-to-end feature

  Specify Y if you want to install end-to-end scheduling and run jobs on Tivoli Workload Scheduler fault-tolerant agents.

- Installation directory

  Specify the (HFS) path where SMP/E has installed the Tivoli Workload Scheduler for z/OS files for UNIX system services that apply the End-to-End enabler feature. This directory is the one containing the bin directory. The default path is /usr/lpp/TWS/V8R2M0.

  The installation directory is created by SMP/E job EQQISMKD and populated by applying the end-to-end feature (JWSZ103).

  This should be mounted read-only on every system in your sysplex.

– Work directory

Specify where the subsystem-specific files are. Replace with a name that uniquely identifies your subsystem. Each subsystem that will use the fault-tolerant workstations must have its own work directory. Only the server and the daily planning batch jobs update the work directory.

This directory is where the end-to-end processes have their working files (Symphony, event files, traces). It should be mounted read/write on every system in your sysplex.

Important: To configure end-to-end scheduling in a sysplex environment successfully, make sure that the work directory is available to all systems in the sysplex. This way, in case of a takeover situation, the new server will be started on a new system in the sysplex, and the server must be able to access the work directory to continue processing.

Hierarchical File System (HFS) cluster: we recommend having dedicated HFS clusters for each end-to-end scheduling environment (end-to-end server started task); that is:

- One HFS cluster for the installation binaries per environment (test, production, and so forth)
- One HFS cluster for the work files per environment (test, production and so forth)

The work HFS clusters should be mounted in read/write mode and the HFS cluster with binaries should be mounted read-only. This is because the working directory is application-specific and contains application-related data. Besides, it makes your backup easier. The size of the cluster depends on the size of the Symphony file and how long you want to keep the stdlist files. We recommend that you allocate 2 GB of space.

– User for OPC address space

This information is used to create the EQQPCS05 sample job that is used to build the directory with the right ownership. In order to run the end-to-end feature correctly, the ownership of the work directory and the files contained in it must be assigned to the same user ID that RACF associates with the server started task. In the User for OPC address space field, specify the RACF user ID used for the Server address space. This is the name specified in the started-procedure table.

– Refresh CP group

This information is used to create the EQQPCS05 sample job used to build the directory with the right ownership. In order to create the new Symphony file, the user ID that is used to run the daily planning batch job must belong to the group that you specify in this field. Make sure that the user ID that is associated with the Server and Controller address spaces (the one specified in the User for OPC address space field) belongs to this group or has this group as a supplementary group.

We defined RACF user ID TWSCE2E to the end-to-end server started task (Figure 15-3). User TWSCE2E belongs to RACF group TWSGRP. Therefore, all users of the RACF group TWSGRP and its supplementary group get access to create the Symphony file and to modify and read other files in the work directory.

Tip: The Refresh CP group field can be used to give access to the HFS file as well as to protect the HFS directory from unauthorized access.

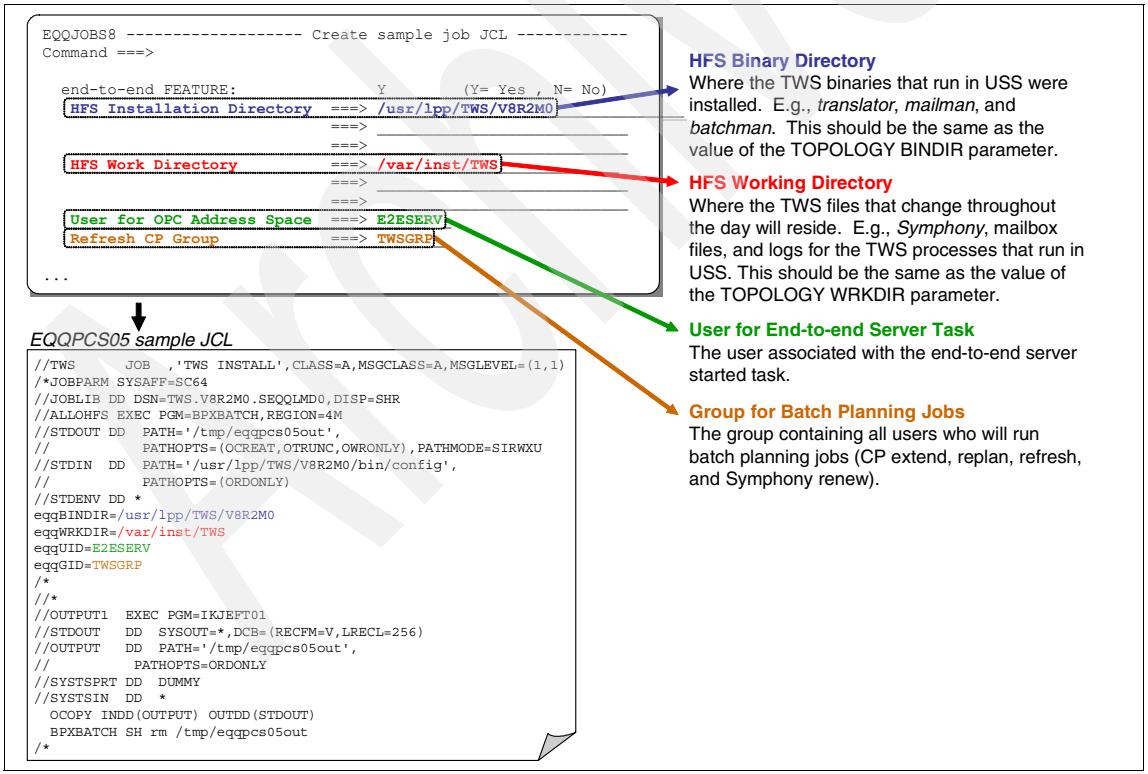


Figure 15-3 Description of the input fields in the EQQJOBS8 panel

3. Press Enter to generate the installation job control language (JCL) jobs. Table 15-1 lists the subset of the sample JCL members created by EQQJOBS that relate to end-to-end scheduling.

Table 15-1 Sample JCL members related to end-to-end scheduling (created by EQQJOBS)

Member	Description
EQQCON	Sample started task procedure for a Tivoli Workload Scheduler for z/OS controller and tracker in the same address space.
EQQCONO	Sample started task procedure for the Tivoli Workload Scheduler for z/OS controller only.
EQQCONP	Sample initial parameters for a Tivoli Workload Scheduler for z/OS controller and tracker in same address space.
EQQCONOP	Sample initial parameters for a Tivoli Workload Scheduler for z/OS controller only.
EQQPCS05	Creates the working directory in HFS used by the end-to-end server task.
EQQPCS06	Allocates data sets necessary to run end-to-end scheduling.
EQQSER	Sample started task procedure for a server task.
EQQSERV	Sample initialization parameters for a server task.

4. EQQJOBS is also used to create batch-job skeletons. That is, skeletons for the batch jobs (such as plan extend, replan, Symphony renew) that you can submit from Tivoli Workload Scheduler for z/OS ISPF panels. To create batch-job skeletons, select option 2 in the EQQJOBS primary panel (see Figure 15-1 on page 382). Make your necessary entries until panel EQQJOBSA appears (Figure 15-4 on page 387).

```
EQQJOBSA -------------- Generate OPC batch-job skeletons ----------------------
Command ===>

 Specify if you want to use the following optional features:

  END TO END FEATURE:                    Y    (Y= Yes ,N= No)
 (To interoperate with TWS
  fault tolerant workstations)

  RESTART AND CLEAN UP (DATA STORE):     N    (Y= Yes ,N= No)
 (To be able to retrieve job log,
  execute dataset clean up actions
  and step restart)

  FORMATTED REPORT OF TRACKLOG EVENTS:   Y    (Y= Yes ,N= No)
    EQQTROUT dsname       ===> TWS.V8R20.*.TRACKLOG_____________________________
    EQQAUDIT output dsn   ===> TWS.V8R20.*.EQQAUDIT.REPORT______________________

Press ENTER to generate OPC batch-job skeletons
```

Figure 15-4 Generate end-to-end skeletons

5. Specify `Y` for the END-TO-END FEATURE if you want to use end-to-end scheduling to run jobs on Tivoli Workload Scheduler fault-tolerant workstations.
6. Press Enter and the skeleton members for daily plan extend, replan, trial plan, long-term plan extend, replan, and trial plan are created with data sets related to end-to-end scheduling. Also, a new member is created (Table 15-2).

Table 15-2 End-to-end skeletons

Member	Description
EQQSYRES	Tivoli Workload Scheduler Symphony renew

15.1.2 Defining Tivoli Workload Scheduler for z/OS subsystems

The subsystem for the Tivoli Workload Scheduler for z/OS controllers (engines) and trackers on the z/OS images (agents) must be defined in the active subsystem-name-table member of SYS1.PARMLIB. It is advisable to install at least two Tivoli Workload Scheduler for z/OS controlling systems, one for testing and one for your production environment.

Note: We recommend that you install the trackers (agents) and the Tivoli Workload Scheduler for z/OS controller (engine) in separate address spaces.

To define the subsystems, update the active IEFSSNnn member in SYS1.PARMLIB. The name of the subsystem initialization module for Tivoli Workload Scheduler for z/OS is EQQINITF. Include records, as in Example 15-1.

Example 15-1 Subsystem definition record (IEFSSNnn member of SYS1.PARMLIB)

```
SUBSYS SUBNAME(subsystem name)      /* TWS for z/OS subsystem */
       INITRTN(EQQINITF)
       INITPARM('maxecsa,F')
```

Note that the subsystem name must be two to four characters: for example, TWSC for the controller subsystem and TWST for the tracker subsystems. Check Chapter 1 for more information on the installation of TWS for z/OS.

15.1.3 Allocate end-to-end data sets

Member EQQPCS06, created by EQQJOBS in your sample job JCL library, allocates the following VSAM and sequential data sets needed for end-to-end scheduling:

- End-to-end script library (EQQSCLIB) for non-centralized script
- End-to-end input and output events data sets (EQQTWSIN, EQQTWSOU)
- Current plan backup copy data set to create Symphony (EQQSCPDS)
- End-to-end centralized script data library (EQQTWSCS)

We now explain the use and allocation of these data sets in more detail.

End-to-end script library (EQQSCLIB)

This script library data set includes members containing the commands or the job definitions for fault-tolerant workstations. It is required in the controller if you want to use the end-to-end scheduling feature. See 15.4.3, “Definition of non-centralized scripts” on page 454 for details about the JOBREC, RECOVERY, and VARSUB statements.

Tip: Do not compress members in this PDS. For example, do not use the ISPF PACK ON command, because Tivoli Workload Scheduler for z/OS does not use ISPF services to read it.

End-to-end input and output data sets

These data sets are required by every Tivoli Workload Scheduler for z/OS address space that uses the end-to-end feature. They record the descriptions of related events with operations running on FTWs and are used by both the end-to-end enabler task and the translator process in the scheduler’s server.

The data sets are device-dependent and can have only primary space allocation. Do not allocate any secondary space. They are automatically formatted by Tivoli Workload Scheduler for z/OS the first time they are used.

Note An SD37 abend code is produced when Tivoli Workload Scheduler for z/OS formats a newly allocated data set. Ignore this error.

EQQTWSIN and EQQTWSOU are wrap-around data sets. In each data set, the header record is used to track the amount of read and write records. To avoid the loss of event records, a writer task will not write any new records until more space is available when all existing records have been read.

The quantity of space that you need to define for each data set requires some attention. Because the two data sets are also used for job log retrieval, the limit for the job log length is half the maximum number of records that can be stored in the input events data set. Two cylinders are sufficient for most installations.

The maximum length of the events logged in those two data sets, including the job logs, is 120 bytes. It is possible to allocate the data sets with a longer logical record length. Using record lengths greater than 120 bytes does not produce either advantages or problems. The maximum allowed value is 32000 bytes; greater values will cause the end-to-end server started task to terminate. In both data sets there must be enough space for at least 1000 events. (The maximum number of job log events is 500.) Use this as a reference if you plan to define a record length greater than 120 bytes. So, when the record length of 120 bytes is used, the space allocation must be at least 1 cylinder. The data sets must be unblocked and the block size must be the same as the logical record length.

A minimum record length of 160 bytes is necessary for the EQQTWSOU data set in order to be able to decide how to build the job name in the Symphony file. (Refer to the TWSJOBNAME parameter in the JTOPTS statement in 15.1.9, “The JTOPTS TWSJOBNAME() parameter” on page 418.)

For good performance, define the data sets on a device with plenty of availability. If you run programs that use the RESERVE macro, try to allocate the data sets on a device that is not, or only slightly, reserved.

Initially, you may need to test your system to get an idea of the number and types of events that are created at your installation. After you have gathered enough information, you can reallocate the data sets. Before you reallocate a data set, ensure that the current plan is entirely up-to-date. You must also stop the end-to-end sender and receiver task on the controller and the translator thread on the server that add this data set.

Tip: Do not move these data sets after they have been allocated. They contain device-dependent information and cannot be copied from one type of device to another, or moved around on the same volume. An end-to-end event data set that is moved will be re-initialized. This causes all events in the data set to be lost. If you have DFHSM or a similar product installed, you should specify that end-to-end event data sets are not migrated or moved.

Current plan backup copy data set (EQQSCPDS)

EQQSCPDS is the current plan backup copy data set that is used to create the Symphony file.

During the creation of the current plan, the SCP data set is used as a CP backup copy for the production of the Symphony file. This VSAM is used when the end-to-end feature is active. It should be allocated with the same size as the CP1/CP2 and NCP VSAM data sets.

End-to-end centralized script data set (EQQTWSCS)

Tivoli Workload Scheduler for z/OS uses the end-to-end centralized script data set to temporarily store a script when it is downloaded from the JOBLIB data set to the agent for its submission.

Set the following attributes for EQQTWSCS:

```
DSNTYPE=LIBRARY,
SPACE=(CYL,(1,1,10)),
DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
```

If you want to use centralized script support when scheduling end-to-end, use the EQQTWSCS DD statement in the controller and server started tasks. The data set must be a partitioned extended-data set.

15.1.4 Create and customize the work directory

To install the end-to-end feature, you must allocate the files that the feature uses. Then, on every Tivoli Workload Scheduler for z/OS controller that will use this feature, run the EQQPCS05 sample to create the directories and files.

The EQQPCS05 sample must be run by a user with one of the following permissions:

- UNIX System Services (USS) user ID (UID) equal to 0
- BPX.SUPERUSER FACILITY class profile in RACF

- UID specified in the JCL in eqqUID and belonging to the group (GID) specified in the JCL in eqqGID

If the GID or the UID has not been specified in EQQJOBS, you can specify them in the STDENV DD before running the EQQPCS05.

The EQQPCS05 job runs a configuration script (named `config`) residing in the installation directory. This configuration script creates a working directory with the right permissions. It also creates several files and directories in this working directory (Figure 15-5).

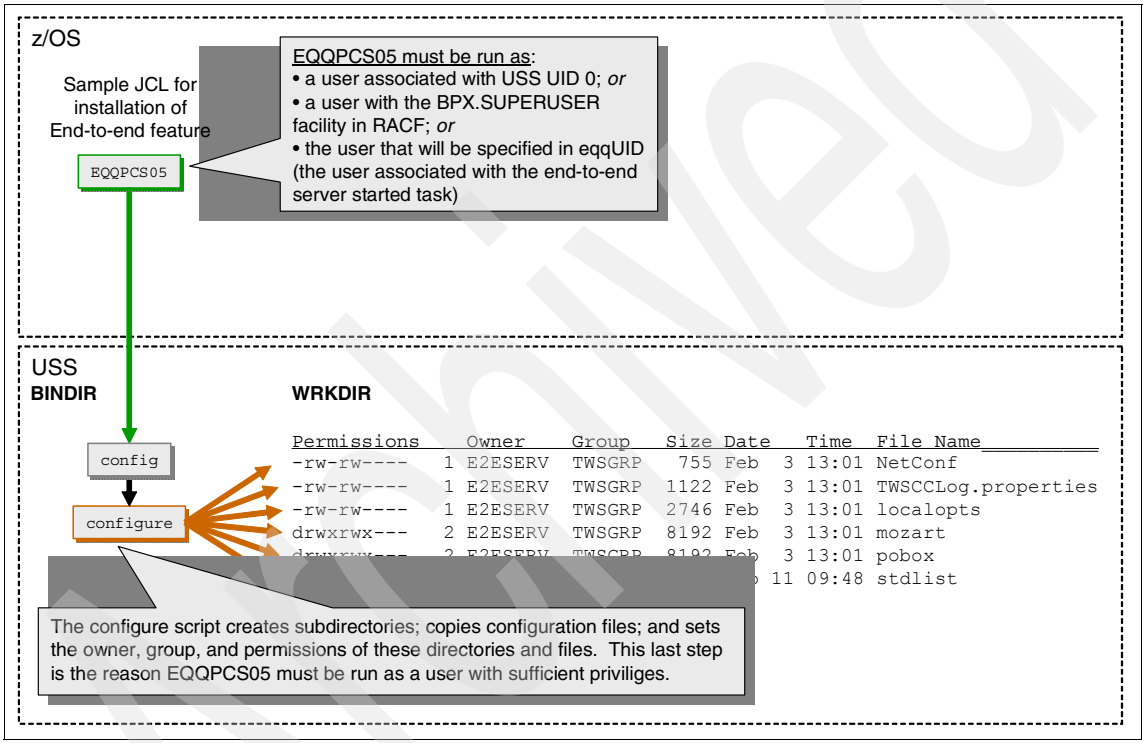


Figure 15-5 EQQPCS05 sample JCL and the configure script

After running EQQPCS05, you can find the following files in the work directory:

- localopts

  Defines the attributes of the local workstation (OPCMASTER) for batchman, mailman, netman, and writer processes and for SSL. Only a subset of these attributes is used by the end-to-end server on z/OS. Refer to *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning,* SC32-1265, for information about customizing this file.

- mozart/globalopts

  Defines the attributes of the IBM Tivoli Workload Scheduler network (OPCMASTER ignores them).

- Netconf

  Netman configuration files.

- TWSCCLOG.properties

  Defines attributes for trace function in the end-to-end server USS.

You will also find the following directories in the work directory:

- mozart
- pobox
- stdlist
- stdlist/logs (contains the log files for USS processes)

Do not touch or delete any of these files or directories, which are created in the work directory by the EQQPCS05 job, unless you are directed to do so, for example in error situations.

Tip: If you execute this job in a sysplex that cannot share the HFS (prior OS/390 V2R9) and get messages such as `cannot create directory`, you may want a closer look on which machine the job really ran. Because without system affinity, every member that has the initiater in the right class started can execute the job so you must add a /*JOBPARM SYSAFF to make sure that the job runs on the system where the work HFS is mounted.

Note that the EQQPCS05 job does *not* define the physical HFS (or z/OS) data set. The EQQPCS05 initiates an existing HFS data set with the necessary files and directories for the end-to-end server started task.

The physical HFS data set can be created with a job that contains an IEFBR14 step, as shown in Example 15-2.

Example 15-2 HFS data set creation

```
//USERHFS EXEC PGM=IEFBR14
//D1      DD DISP=(,CATLG),DSNTYPE=HFS,
//           SPACE=(CYL,(prispace,secspace,1)),
//           DSN=OMVS.TWS820.TWSCE2E.HFS
```

Allocate the HFS work data set with enough space for your end-to-end server started task. In most installations, 2 GB disk space is enough.

15.1.5 Create started task procedures

Perform this task for a Tivoli Workload Scheduler for z/OS tracker (agent), controller (engine), and server started task. You must define a started task procedure or batch job for each Tivoli Workload Scheduler for z/OS address space. (For more information, see Chapter 3, “The started tasks” on page 69.)

The EQQJOBS dialog generates several members in the output sample library that you specified when running the EQQJOBS installation aid program. These members contain started task JCL that is tailored with the values you entered in the EQQJOBS dialog. Tailor these members further, according to the data sets you require. (See Table 15-1 on page 386.)

Because the end-to-end server started task uses TCP/IP communication, you should take the following steps.

1. Modify the JCL of EQQSER in the following way:
   a. Make sure that the end-to-end server started task has access to the C runtime libraries, either as STEPLIB (include the CEE.SCEERUN in the STEPLIB concatenation) or by LINKLIST (the CEE.SCEERUN is in the LINKLIST concatenation).
   b. If you have multiple TCP/IP stacks, or if the name you used for the procedure that started up the TCP/IP address space was not the default (TCPIP), change the end-to-end server started task procedure to include the SYSTCPD DD card to point to a data set containing the TCPIPJOBNAME parameter. The standard method to determine the connecting TCP/IP image is:
      i. Connect to the TCP/IP specified by TCPIPJOBNAME in the active TCPIP.DATA.
      ii. Locate TCPIP.DATA using the SYSTCPD DD card.

      You can also use the end-to-end server TOPOLOGY TCPIPJOBNAME() parameter to specify the TCP/IP started task name that is used by the end-to-end server. This parameter can be used if you have multiple TCP/IP stacks or if the TCP/IP started task name is different from TCPIP.
2. You must have a server started task to handle end-to-end scheduling. You can use the same server to communicate with the Job Scheduling Console. In fact, the server can also handle APPC communication if configured to this.
3. In Tivoli Workload Scheduler for z/OS 8.2, the type of communication that should be handled by the server started task is defined in the new SERVOPTS PROTOCOL() parameter.

In the PROTOCOL() parameter, you can specify any combination of:

- APPC: The server should handle APPC communication.

- JSC: The server should handle JSC communication.
- E2E: The server should handle end-to-end communication.

Recommendations: The Tivoli Workload Scheduler for z/OS controller and end-to-end server use TCP/IP services, so you must define a USS segment for the controller and end-to-end server started task user IDs. No special authorization is necessary; it is only required to be defined in USS with any user ID.

Even though it is possible to have one server started task handling end-to-end scheduling, JSC communication, and even APPC communication as well, we recommend having a server started task dedicated for end-to-end scheduling (SERVOPTS PROTOCOL(E2E)). This has the advantage that you do not have to stop the whole server processes if the JSC Server must be restarted.

The server started task is important for handling JSC and end-to-end communication. We recommend setting the end-to-end and JSC Server started tasks as non-swappable and giving it at least the same dispatching priority as the Tivoli Workload Scheduler for z/OS controller (engine).

The Tivoli Workload Scheduler for z/OS controller uses the end-to-end server to communicate events to the FTAs. The end-to-end server will start multiple tasks and processes using the UNIX System Services.

15.1.6 Initialization statements for Tivoli Workload Scheduler for z/OS end-to-end scheduling

Initialization statements for end-to-end scheduling fit into two categories:

- Statements used to configure the Tivoli Workload Scheduler for z/OS controller (engine) and end-to-end server:
  - OPCOPTS and TPLGYPRM statements for the controller
  - SERVOPTS statement for the end-to-end server
- Statements used to define the end-to-end topology (the network topology for the distributed Tivoli Workload Scheduler network). The end-to-end topology statements fall into two categories:
  - Topology statements used to initialize the end-to-end server environment in USS on the mainframe (the TOPOLOGY statement)

– Statements used to describe the distributed Tivoli Workload Scheduler network and the responsibilities for the different Tivoli Workload Scheduler agents in this network (the DOMREC, CPUREC, and USRREC statements)

These statements are used by the end-to-end server and the plan extend, plan replan, and Symphony renew batch jobs. The batch jobs use the information when the Symphony file is created.

See 15.1.7, “Initialization statements used to describe the topology” on page 403.

We go through each initialization statement in detail and give you an example of how a distributed Tivoli Workload Scheduler network can be reflected in Tivoli Workload Scheduler for z/OS using the topology statements.

Table 15-3 Initialization members related to end-to-end scheduling

Initialization member	Description
TPLGYSRV	Activates end-to-end in the Tivoli Workload Scheduler for z/OS controller.
TPLGYPRM	Activates end-to-end in the Tivoli Workload Scheduler for server and batch jobs (plan jobs).
TOPOLOGY	Specifies all statements for end-to-end.
DOMREC	Defines domains in a distributed Tivoli Workload Scheduler network.
CPUREC	Defines agents in a Tivoli Workload Scheduler distributed network.
USRREC	Specifies user ID and password for NT users.

Find more information in *Tivoli Workload Scheduler for z/OS Customization and Tuning,* SH19-4544.

Figure 15-6 on page 396 illustrates the relationship between the initialization statements and members related to end-to-end scheduling.

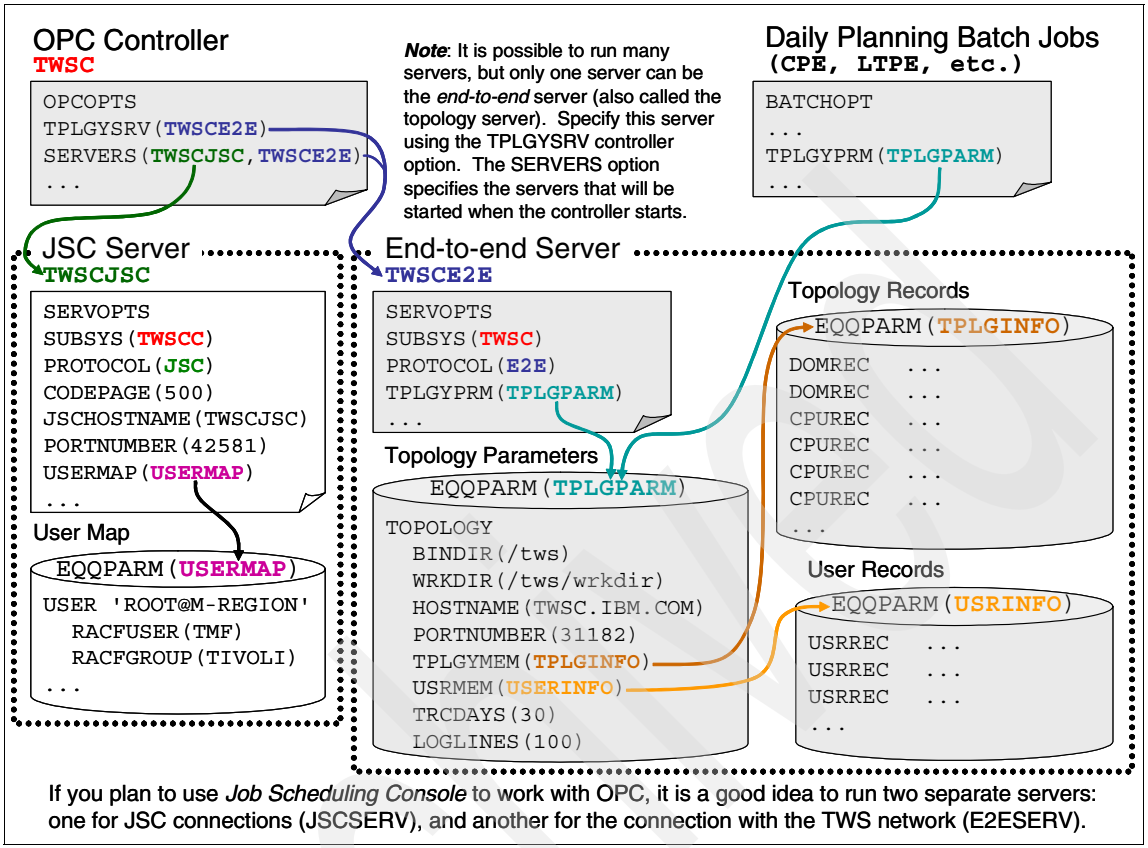


Figure 15-6 Relationship between end-to-end initialization statements and members

In the following sections, we cover the different initialization statements and members and describe their meaning and usage one by one. Refer to Figure 15-6 when reading these sections.

OPCOPTS TPLGYSRV(server_name)

Specify this keyword if you want to activate the end-to-end feature in the Tivoli Workload Scheduler for z/OS (OPC) controller (engine).

Activates the end-to-end feature in the controller. If you specify this keyword, the IBM Tivoli Workload Scheduler Enabler task is started. The specified server_name is that of the end-to-end server that handles the events to and from the FTAs. Only one server can handle events to and from the FTAs.

This keyword is defined in OPCOPTS.

Tip: If you want to let the Tivoli Workload Scheduler for z/OS controller start and stop the end-to-end server, use the `servers` keyword in OPCOPTS parmlib member. (See Figure 15-6 on page 396.)

SERVOPTS TPLGYPRM(member name/TPLGPARM)

The SERVOPTS statement is the first statement read by the end-to-end server started task. In the SERVOPTS, you specify different initialization options for the server started task, such as:

- The name of the Tivoli Workload Scheduler for z/OS controller that the server should communicate with (serve). The name is specified with the SUBSYS() keyword.
- The type of protocol. The PROTOCOL() keyword is used to specify the type of communication used by the server.

  In Tivoli Workload Scheduler for z/OS 8.2, you can specify any combination of the following values separated by comma: E2E, JSC, APPC.

  Note: With Tivoli Workload Scheduler for z/OS 8.2, the TCPIP value has been replaced by the combination of the E2E and JSC values, but the TCPIP value is still allowed for backward compatibility.

- The TPLGYPRM() parameter is used to define the member name of the member in parmlib with the TOPOLOGY definitions for the distributed Tivoli Workload Scheduler network.

  The TPLGYPRM() parameter *must* be specified if PROTOCOL(E2E) is specified.

See Figure 15-6 on page 396 for an example of the required SERVOPTS parameters for an end-to-end server (`TWSCE2E` in Figure 15-6 on page 396).

TPLGYPRM(member name/TPLGPARM) in BATCHOPT

It is important to remember to add the TPLGYPRM() parameter to the BATCHOPT initialization statement that is used by the Tivoli Workload Scheduler for z/OS planning jobs (trial plan extend, plan extend, plan replan) and Symphony renew.

If the TPLGYPRM() parameter is not specified in the BATCHOP initialization statement that is used by the plan jobs, no Symphony file will be created and no jobs will run in the distributed Tivoli Workload Scheduler network.

Figure 15-6 on page 396 shows an example of how to specify the TPLGYPRM() parameter in the BATCHOPT initialization statement.

Note: Topology definitions in TPLGYPRM() in the BATCHOPT initialization statement are read and verified by the trial plan extend job in Tivoli Workload Scheduler for z/OS. Thus the trial plan extend job can be used to verify the TOPLOGY definitions such as DOMREC, CPUREC, and USRREC for syntax errors or logical errors before the plan extend or plan replan job is executed. *The trial plan extend job does not create a new Symphony file because it does not update the current plan in Tivoli Workload Scheduler for z/OS.*

TOPOLOGY statement

This statement includes all of the parameters that are related to the end-to-end feature. TOPOLOGY is defined in the member of the EQQPARM library as specified by the TPLGYPRM parameter in the BATCHOPT and SERVOPTS statements. Figure 15-8 on page 404 shows the syntax of the topology member.

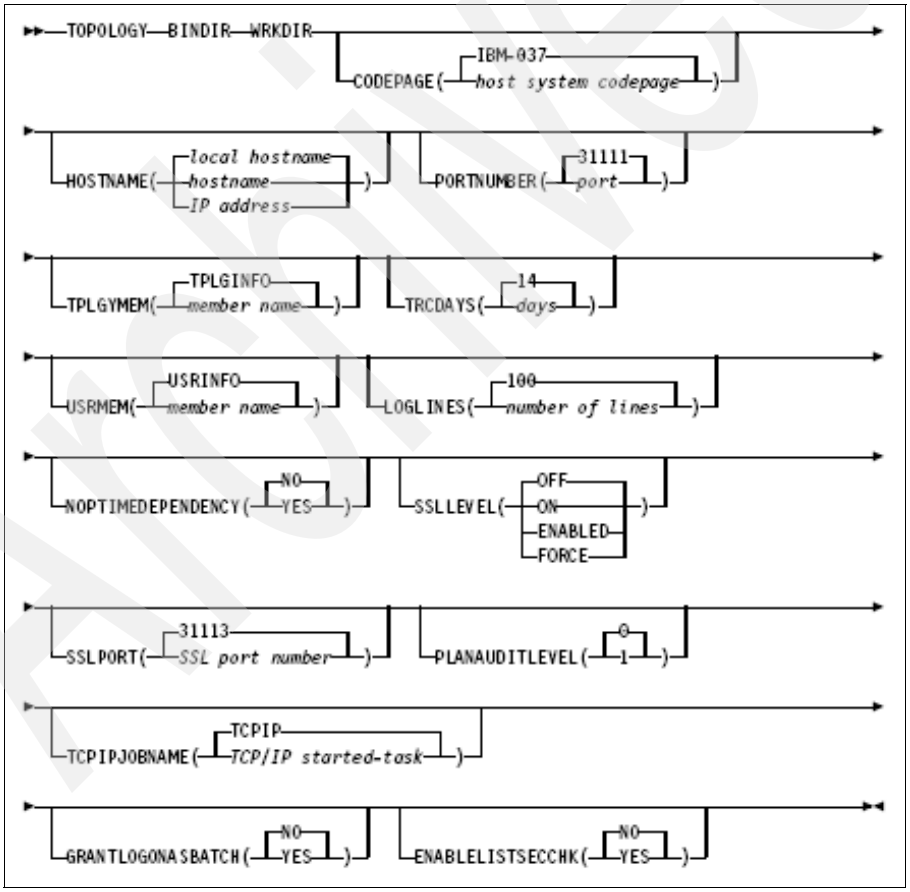


Figure 15-7 The statements that can be specified in the topology member

Description of the topology statements

The following sections describe the topology parameters.

BINDIR(directory name)

Specifies the name of the base file system (HFS or zOS) directory where binaries, catalogs, and the other files are installed and shared among subsystems.

The specified directory must be the same as the directory where the binaries are, without the final `bin`. (If the binaries are installed in /usr/lpp/TWS/V8R2M0/bin and the catalogs are in /usr/lpp/TWS/V8R2M0/catalog/C, the directory must be specified in the BINDIR keyword as follows: /usr/lpp/TWS/V8R2M0.)

CODEPAGE(host system codepage/IBM-037)

Specifies the name of the host code page; applies to the end-to-end feature. The value is used by the input translator to convert data received from Tivoli Workload Scheduler domain managers at the first level from UTF-8 format to EBCIDIC format. You can provide the `IBM – xxx` value, where *xxx* is the EBCDIC code page. The default value, IBM – 037, defines the EBCDIC code page for US English, Portuguese, and Canadian French.

For a complete list of available code pages, refer to *Tivoli Workload Scheduler for z/OS Customization and Tuning,* SH19-4544.

ENABLELISTSECCHK(YES/NO)

This security option controls the ability to list objects in the plan on an FTA using conman and the JSC. Put simply, this option determines whether conman and the Tivoli Workload Scheduler connector programs will check the Tivoli Workload Scheduler Security file before allowing the user to list objects in the plan.

If set to YES, objects in the plan are shown to the user only if the user has been granted the list permission in the Security file. If set to NO, all users will be able to list objects in the plan on FTAs, regardless of whether list access is granted in the Security file. The default value is NO. Change the value to YES if you want to check for the list permission in the security file.

GRANTLOGONASBATCH(YES/NO)

This is *only* for jobs running on Windows platforms. If set to YES, the logon users for Windows jobs are automatically granted the right to log on as batch job. If set to NO or omitted, the right must be granted manually to each user or group. The right cannot be granted automatically for users running jobs on a backup domain controller, so you must grant those rights manually.

HOSTNAME(host name /IP address/ local host name)

Specifies the host name or the IP address used by the server in the end-to-end environment. The default is the host name returned by the operating system.

If you change the value, you also must restart the Tivoli Workload Scheduler for z/OS server and renew the Symphony file.

You can define a virtual IP address for each server of the active controller and the standby controllers. If you use a dynamic virtual IP address in a sysplex environment, when the active controller fails and the standby controller takes over the communication, the FTAs automatically switch the communication to the server of the standby controller.

To change the HOSTNAME of a server, perform the following actions:

1. Set the nm ipvalidate keyword to `off` in the localopts file on the first-level domain managers.
2. Change the HOSTNAME value of the server using the TOPOLOGY statement.
3. Restart the server with the new HOSTNAME value.
4. Renew the Symphony file.
5. If the renewal ends successfully, you can set the ipvalidate to `full` on the first-level domain managers.

LOGLINES(number of lines/100)

Specifies the maximum number of lines that the job log retriever returns for a single job log. The default value is 100. In all cases, the job log retriever does not return more than half of the number of records that exist in the input queue.

If the job log retriever does not return all of the job log lines because there are more lines than the LOGLINES() number of lines, a notice similar to this appears in the retrieved job log output:

```
*** nnn lines have been discarded. Final part of Joblog ...  ******
```

The line specifies the number (*nnn*) of job log lines not displayed, between the first lines and the last lines of the job log.

NOPTIMEDEPENDENCY(YES/NO)

With this option, you can change the behavior of noped operations that are defined on fault-tolerant workstations and have the Centralized Script option set to N. In fact, Tivoli Workload Scheduler for z/OS completes the noped operations without waiting for the time dependency resolution: With this option set to YES, the operation can be completed in the current plan after the time dependency has been resolved. The default value is NO.

Note: This statement is introduced by APAR PQ84233.

PLANAUDITLEVEL(0/1)

Enables or disables plan auditing for FTAs. Each Tivoli Workload Scheduler workstation maintains its own log. Valid values are 0 to disable plan auditing and 1 to activate plan auditing. Auditing information is logged to a flat file in the TWShome/audit/plan directory. Only actions, not the success or failure of any action are logged in the auditing file. If you change the value, you must restart the Tivoli Workload Scheduler for z/OS server and renew the Symphony file.

PORTNUMBER(port/31111)

Defines the TCP/IP port number that is used by the server to communicate with the FTAs. This value has to be different from that specified in the SERVOPTS member. The default value is 31111, and accepted values are from 0 to 65535.

If you change the value, you must restart the Tivoli Workload Scheduler for z/OS server and renew the Symphony file.

Important: The port number must be unique within a Tivoli Workload Scheduler network.

SSLLEVEL(ON/OFF/ENABLED/FORCE)

Defines the type of SSL authentication for the end-to-end server (OPCMASTER workstation). It must have one of the following values:

ON The server uses SSL authentication only if another workstation requires it.

OFF (default value) The server does not support SSL authentication for its connections.

ENABLED The server uses SSL authentication only if another workstation requires it.

FORCE The server uses SSL authentication for all of its connections. It refuses any incoming connection if it is not SSL.

If you change the value, you must restart the Tivoli Workload Scheduler for z/OS server and renew the Symphony file.

SSLPORT(SSL port number/31113)

Defines the port used to listen for incoming SSL connections on the server. It substitutes the value of nm SSL port in the localopts file, activating SSL support on the server. If SSLLEVEL is specified and SSLPORT is missing, 31113 is used as the default value. If SSLLEVEL is not specified, the default value of this parameter is 0 on the server, which indicates that no SSL authentication is required.

If you change the value, you must restart the Tivoli Workload Scheduler for z/OS server and renew the Symphony file.

TCPIPJOBNAME(TCP/IP started-task name/TCPIP)
Specifies the TCP/IP started-task name used by the server. Set this keyword when you have multiple TCP/IP stacks or a TCP/IP started task with a name different from TCPIP. You can specify a name from one to eight alphanumeric or national characters, where the first character is alphabetic or national.

TPLGYMEM(member name/TPLGINFO)
Specifies the PARMLIB member where the domain (DOMREC) and workstation (CPUREC) definition specific to the end-to-end are. Default value is TPLGINFO.

If you change the value, you must restart the Tivoli Workload Scheduler for z/OS server and renew the Symphony file.

TRCDAYS(days/14)
Specifies the number of days the trace files and file in the stdlist directory are kept before being deleted. Every day the USS code creates the new stdlist directory to contain the logs for the day. All log directories that are older than the number of days specified in TRCDAYS() are deleted automatically. The default value is 14. Specify 0 if you do not want the trace files to be deleted.

Recommendation: Monitor the size of your working directory (that is, the size of the HFS cluster with work files) to prevent the HFS cluster from becoming full. The trace files and files in the stdlist directory contain internal logging information and Tivoli Workload Scheduler messages that may be useful for troubleshooting. You should consider deleting them on a regular interval using the TRCDAYS() parameter.

USRMEM(member name/USRINFO)
Specifies the PARMLIB member where the user definitions are. This keyword is optional *except* if you are going to schedule jobs on Windows operating systems, in which case, it is *required*. The default value is USRINFO.

If you change the value, you must restart the Tivoli Workload Scheduler for z/OS server and renew the Symphony file.

WRKDIR(directory name)
Specifies the location of the working files for an end-to-end server started task. Each Tivoli Workload Scheduler for z/OS end-to-end server must have its own WRKDIR.

ENABLESWITCHFT(Y/N)
New parameter (not shown in Figure 15-7 on page 398) that was introduced in FixPack 04 for Tivoli Workload Scheduler and APAR PQ81120 for Tivoli Workload Scheduler.

It is used to activated the enhanced fault-tolerant mechanism on domain managers. The default is N (the enhanced fault-tolerant mechanism is not activated). For more information, check the FaultTolerantSwitch.README.pdf file delivered with FixPack 04 for Tivoli Workload Scheduler.

15.1.7 Initialization statements used to describe the topology

With the last three parameters listed in Table 15-3 on page 395, DOMREC, CPUREC, and USRREC, you define the topology of the distributed Tivoli Workload Scheduler network in Tivoli Workload Scheduler for z/OS. The defined topology is used by the plan extend, replan, and Symphony renew batch jobs when creating the Symphony file for the distributed Tivoli Workload Scheduler network.

Figure 15-8 shows how the distributed Tivoli Workload Scheduler topology is described using CPUREC and DOMREC initialization statements for the Tivoli Workload Scheduler for z/OS server and plan programs. The Tivoli Workload Scheduler for z/OS fault-tolerant workstations are mapped to physical Tivoli Workload Scheduler agents or workstations using the CPUREC statement. The DOMREC statement is used to describe the domain topology in the distributed Tivoli Workload Scheduler network.

Figure 15-8 does *not* depict the USRREC parameters.

The MASTERDM domain is predefined in Tivoli Workload Scheduler for z/OS. It is not necessary to specify a DOMREC parameter for the MASTERDM domain.

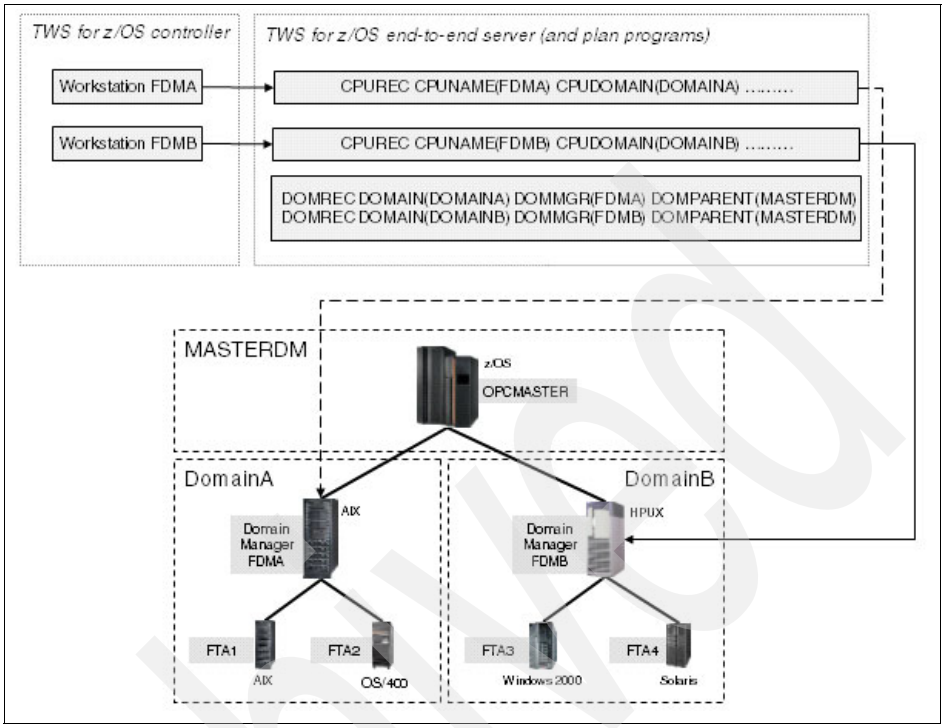


Figure 15-8 The topology definitions for server and plan programs

We now walk through the DOMREC, CPUREC, and USRREC statements.

DOMREC statement

This statement begins a domain definition. You must specify one DOMREC for each domain in the Tivoli Workload Scheduler network, with the exception of the master domain.

The domain name used for the master domain is MASTERDM. The master domain consists of the controller, which acts as the master domain manager. The CPU name used for the master domain manager is OPCMASTER.

You must specify at least one domain, child of MASTERDM, where the domain manager is a fault-tolerant agent. If you do not define this domain, Tivoli Workload Scheduler for z/OS tries to find a domain definition that can function as a child of the master domain.

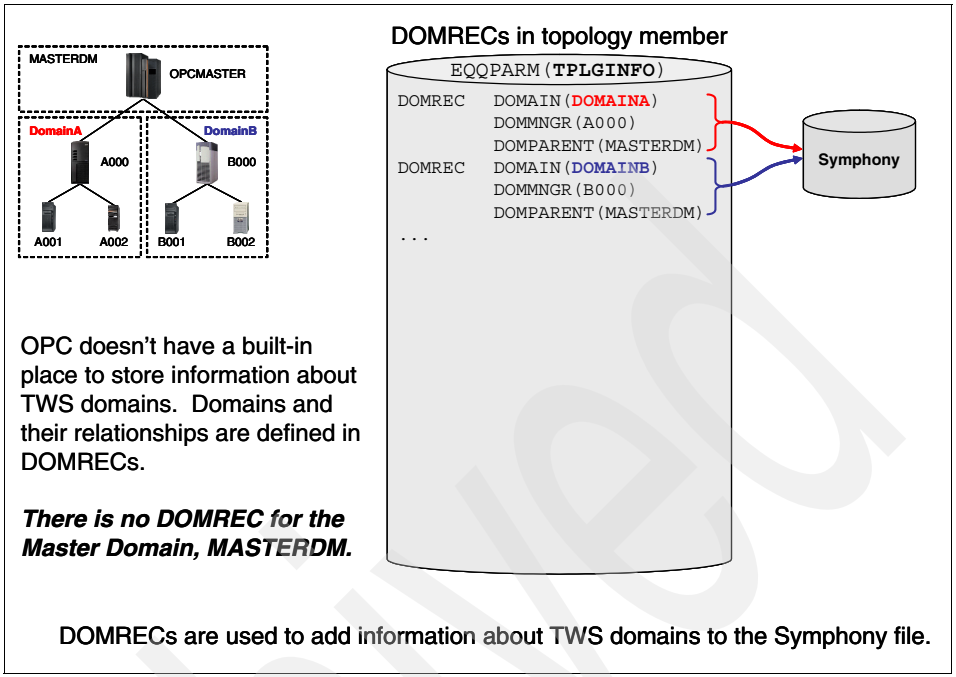


Figure 15-9 Example of two DOMREC statements for a network with two domains

DOMREC is defined in the member of the EQQPARM library that is specified by the TPLGYMEM keyword in the TOPOLOGY statement (see Figure 15-6 on page 396 and Figure 15-9).

Figure 15-10 illustrates the DOMREC syntax.

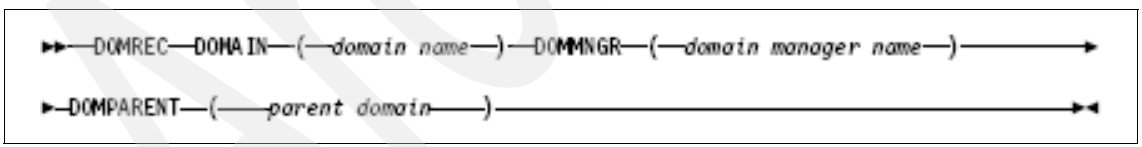


Figure 15-10 Syntax for the DOMREC statement

DOMAIN(domain name)

The name of the domain, consisting of up to 16 characters starting with a letter. It can contain dashes and underscores.

DOMMNGR(domain manager name)

The Tivoli Workload Scheduler workstation name of the domain manager. It must be a fault-tolerant agent running in full status mode.

DOMPARENT(parent domain)
The name of the parent domain.

CPUREC statement

This statement begins a Tivoli Workload Scheduler workstation (CPU) definition. You must specify one CPUREC for each workstation in the Tivoli Workload Scheduler network, with the exception of the controller that acts as master domain manager. You must provide a definition for each workstation of Tivoli Workload Scheduler for z/OS that is defined in the database as a Tivoli Workload Scheduler fault-tolerant workstation.

CPUREC is defined in the member of the EQQPARM library that is specified by the TPLGYMEM keyword in the TOPOLOGY statement (see Figure 15-6 on page 396 and Figure 15-11).

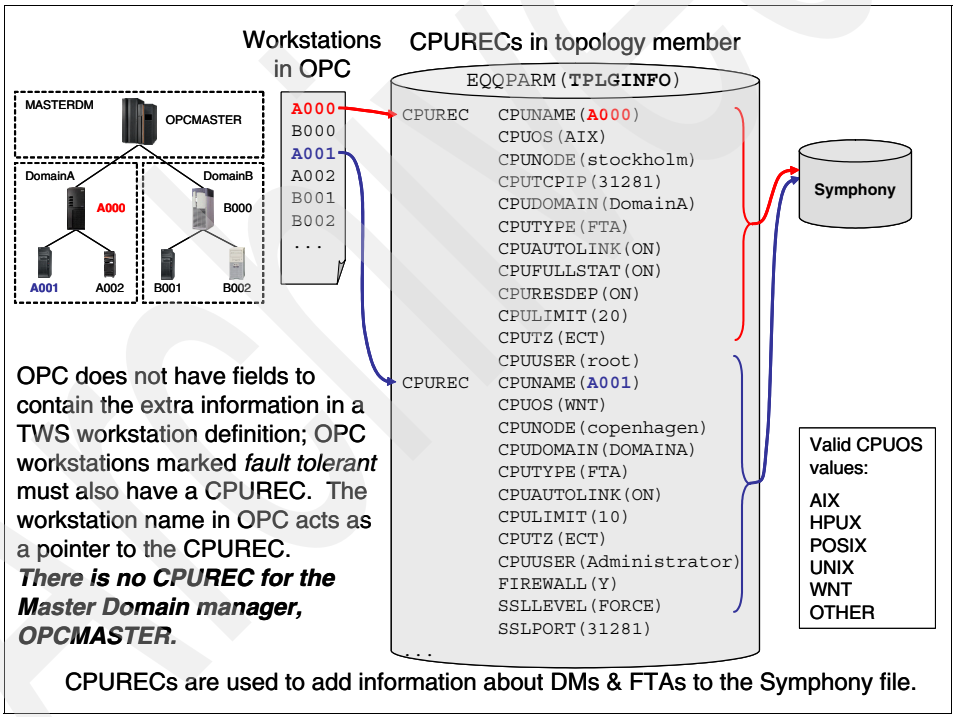


Figure 15-11 Example of two CPUREC statements for two workstations

Figure 15-12 illustrates the CPUREC syntax.

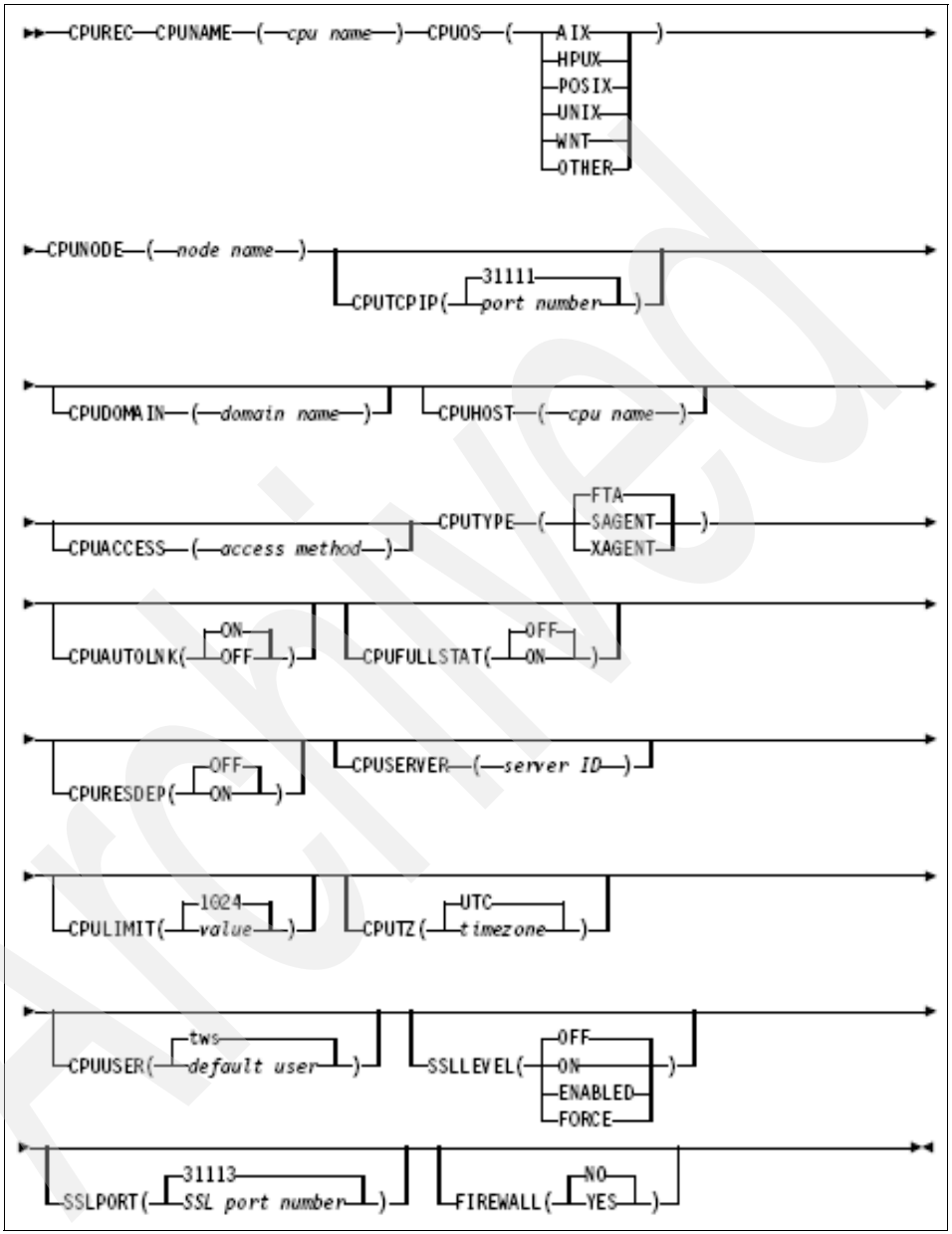


Figure 15-12 Syntax for the CPUREC statement

CPUNAME(cpu name)
The name of the Tivoli Workload Scheduler workstation, consisting of up to four alphanumerical characters, starting with a letter.

CPUOS(operating system)
The host CPU operating system related to the Tivoli Workload Scheduler workstation. The valid entries are AIX, HPUX, POSIX, UNIX, WNT, and OTHER.

CPUNODE(node name)
The node name or the IP address of the CPU. Fully-qualified domain names up to 52 characters are accepted.

CPUTCPIP(port number/ 31111)
The TCP port number of netman on this CPU. It comprises five numbers and, if omitted, uses the default value, 31111.

CPUDOMAIN(domain name)
The name of the Tivoli Workload Scheduler domain of the CPU.

CPUHOST(cpu name)
The name of the host CPU of the agent. It is required for standard and extended agents. The host is the Tivoli Workload Scheduler CPU with which the standard or extended agent communicates and where its access method resides.

Note: The host cannot be another standard or extended agent.

CPUACCESS(access method)
The name of the access method. It is valid for extended agents and must be the name of a file that resides in the Tivoli Workload Scheduler <home>/methods directory on the host CPU of the agent.

CPUTYPE(SAGENT/ XAGENT/ FTA)
The CPU type specified as one of these:

FTA	(default) Fault-tolerant agent, including domain managers and backup domain managers.
SAGENT	Standard agent
XAGENT	Extended agent

Note: If the extended-agent workstation is manually set to Link, Unlink, Active, or Offline, the command is sent to its host CPU.

CPUAUTOLNK(OFF/ON)

Autolink is most effective during the initial start-up sequence of each plan. Then a new Symphony file is created and all workstations are stopped and restarted.

- For a fault-tolerant agent or standard agent, specify ON so that, when the domain manager starts, it sends the new production control file (Symphony) to start the agent and open communication with it.

  For the domain manager, specify ON so that when the agents start they open communication with the domain manager.

- Specify OFF to initialize an agent when you submit a link command manually from the Tivoli Workload Scheduler for z/OS Modify Current Plan ISPF dialogs or from the Job Scheduling Console.

> **Note:** If the X-agent workstation is manually set to Link, Unlink, Active, or Offline, the command is sent to its host CPU.

CPUFULLSTAT(ON/OFF)

This applies only to fault-tolerant agents. If you specify OFF for a domain manager, the value is forced to ON.

- Specify ON for the link from the domain manager to operate in Full Status mode. In this mode, the agent is kept updated about the status of jobs and job streams that are running on other workstations in the network.
- Specify OFF for the agent to receive status information only about the jobs and schedules on other workstations that affect its own jobs and schedules. This can improve the performance by reducing network traffic.

To keep the production control file for an agent at the same level of detail as its domain manager, set CPUFULLSTAT and CPURESDEP to ON. *Always set these modes to ON for backup domain managers.*

You should also be aware of the new TOPOLOGY ENABLESWITCHFT() parameter described in “ENABLESWITCHFT(Y/N)” on page 402.

CPURESDEP(ON/OFF)

This applies only to fault-tolerant agents. If you specify OFF for a domain manager, the value is forced to ON.

- Specify ON to have the agent’s production control process operate in Resolve All Dependencies mode. In this mode, the agent tracks dependencies for all of its jobs and schedules, including those running on other CPUs.

> **Note:** CPUFULLSTAT must also be ON so that the agent is informed about the activity on other workstations.

- Specify OFF if you want the agent to track dependencies only for its own jobs and schedules. This reduces CPU usage by limiting processing overhead.

To keep the production control file for an agent at the same level of detail as its domain manager, set CPUFULLSTAT and CPURESDEP to ON. *Always set these modes to ON for backup domain managers.*

You should also be aware of the new TOPOLOGY ENABLESWITCHFT() parameter that is described in “ENABLESWITCHFT(Y/N)” on page 402.

CPUSERVER(server ID)

This applies only to fault-tolerant and standard agents. Omit this option for domain managers.

This keyword can be a letter or a number (A-Z or 0-9) and identifies a server (mailman) process on the domain manager that sends messages to the agent. The IDs are unique to each domain manager, so you can use the same IDs for agents in different domains without conflict. If more than 36 server IDs are required in a domain, consider dividing it into two or more domains.

If a server ID is not specified, messages to a fault-tolerant or standard agent are handled by a single mailman process on the domain manager. Entering a server ID causes the domain manager to create an additional mailman process. The same server ID can be used for multiple agents. The use of servers reduces the time that is required to initialize agents and generally improves the timeliness of messages.

Notes on multiple mailman processes:

- When setting up multiple mailman processes, do not forget that each mailman server process uses extra CPU resources on the workstation on which it is created, so be careful not to create excessive mailman processes on low-end domain managers. In most of the cases, using extra domain managers is a better choice than configuring extra mailman processes.
- Cases when use of extra mailman processes might be beneficial include:
  - Important FTAs that run mission-critical jobs.
  - Slow-initializing FTAs that are at the other end of a slow link. (If you have more than a couple of workstations over a slow link connection to the OPCMASTER, a better idea is to place a remote domain manager to serve these workstations.)
- If you have unstable workstations in the network, do not put them under the same mailman server ID with your critical servers.

Figure 15-13 shows an example of CPUSERVER() use in which one mailman process on domain manager FDMA has to handle all outbound communication with the five FTAs (FTA1 to FTA5) if these workstations (CPUs) are defined without the CPUSERVER() parameter. If FTA1 and FTA2 are defined with CPUSERVER(A), and FTA3 and FTA4 are defined with CPUSERVER(1), the domain manager FDMA will start two new mailman processes for these two server IDs (A and 1).

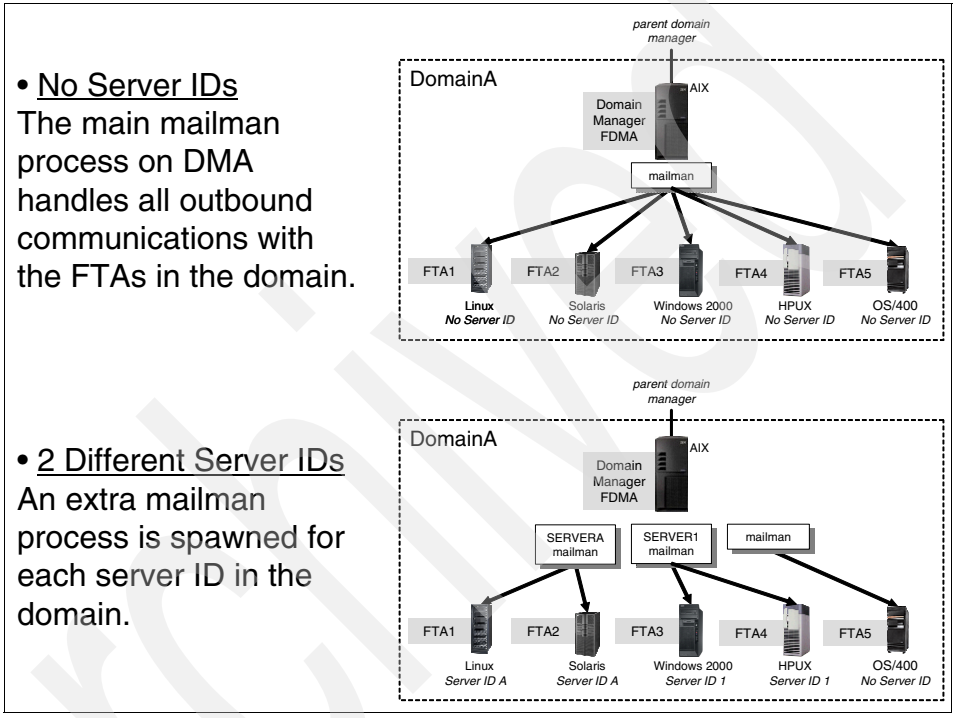


Figure 15-13 Use of CPUSERVER() IDs to start extra mailman processes

CPULIMIT(value/1024)

Specifies the number of jobs that can run at the same time in a CPU. The default value is 1024. The accepted values are integers from 0 to 1024. If you specify 0, no jobs are launched on the workstation.

CPUTZ(timezone/UTC)

Specifies the local time zone of the FTA. It must match the time zone of the operating system in which the FTA runs. For a complete list of valid time zones, refer to the appendix of the *IBM Tivoli Workload Scheduler Reference Guide*, SC32-1274.

If the time zone does not match that of the agent, the message AWSBHT128I displays in the log file of the FTA. The default is UTC (universal coordinated time).

To avoid inconsistency between the local date and time of the jobs and of the Symphony creation, use the CPUTZ keyword to set the local time zone of the fault-tolerant workstation. If the Symphony creation date is later than the current local date of the FTW, Symphony is not processed.

In the end-to-end environment, time zones are disabled by default when installing or upgrading Tivoli Workload Scheduler for z/OS. If the CPUTZ keyword is not specified, time zones are disabled. For additional information about how to set the time zone in an end-to-end network, see the *IBM Tivoli Workload Scheduler Planning and Installation Guide,* SC32-1273.

CPUUSER(default user/tws)
Specifies the default user for the workstation. The maximum length is 47 characters. The default value is tws.

The value of this option is used only if you have not defined the user in the JOBUSR option of the SCRPTLIB JOBREC statement or supply it with the Tivoli Workload Scheduler for z/OS job submit exit EQQUX001 for centralized scripts.

SSLLEVEL(ON/OFF/ENABLED/FORCE)
Must have one of the following values:

ON The workstation uses SSL authentication when it connects with its domain manager. The domain manager uses the SSL authentication when it connects with a domain manager of a parent domain. However, it refuses any incoming connection from its domain manager if the connection does not use the SSL authentication.

OFF (default) The workstation does not support SSL authentication for its connections.

ENABLED The workstation uses SSL authentication only if another workstation requires it.

FORCE The workstation uses SSL authentication for all of its connections. It refuses any incoming connection if it is not SSL.

If this attribute is set to OFF or omitted, the workstation is not intended to be configured for SSL. In this case, any value for SSLPORT (see below) will be ignored. You should also set the nm ssl port local option to 0 (in the localopts file) to be sure that this port is not opened by netman.

SSLPORT(SSL port number|/31113)

Defines the port used to listen for incoming SSL connections. This value must match the one defined in the *nm SSL port* local option (in the localopts file) of the workstation (the server with Tivoli Workload Scheduler installed). It must be different from the *nm port* local option (in the localopts file) that defines the port used for normal communications. If SSLLEVEL is specified but SSLPORT is missing, 31113 is used as the default value. If not even SSLLEVEL is specified, the default value of this parameter is 0 on FTWs, which indicates that no SSL authentication is required.

FIREWALL(YES/NO)

Specifies whether the communication between a workstation and its domain manager must cross a firewall. If you set the FIREWALL keyword for a workstation to YES, it means that a firewall exists between that particular workstation and its domain manager, and that the link between the domain manager and the workstation (which can be another domain manager itself) is the only link that is allowed between the respective domains. Also, for all workstations having this option set to YES, the commands to start (**`start workstation`**) or stop (**`stop workstation`**) the workstation or to get the standard list (**`showjobs`**) are transmitted through the domain hierarchy instead of opening a direct connection between the master (or domain manager) and the workstation. The default value for FIREWALL is NO, meaning that there is no firewall boundary between the workstation and its domain manager.

To specify that an extended agent is behind a firewall, set the FIREWA**LL** keyword for the host workstation. The host workstation is the Tivoli Workload Scheduler workstation with which the extended agent communicates and where its access method resides.

USRREC statement

This statement defines the passwords for the users who need to schedule jobs to run on Windows workstations.

USRREC is defined in the member of the EQQPARM library as specified by the USERMEM keyword in the TOPOLOGY statement. (See Figure 15-6 on page 396 and Figure 15-15 on page 415.)

Figure 15-14 illustrates the USRREC syntax.

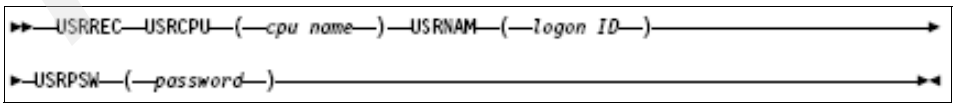


Figure 15-14 Syntax for the USRREC statement

USRCPU(cpuname)

The Tivoli Workload Scheduler workstation on which the user can launch jobs. It consists of four alphanumerical characters, starting with a letter. It is valid only on Windows workstations.

USRNAM(logon ID)

The user name of a Windows workstation. It can include a domain name and can consist of 47 characters.

Windows user names are case sensitive. The user must be able to log on to the computer on which Tivoli Workload Scheduler has launched jobs, and must also be authorized to log on as `batch`.

If the user name is not unique in Windows, it is considered to be either a local user, a domain user, or a trusted domain user, in that order.

USRPWD(password)

The user password for the user of a Windows workstation (Figure 15-15 on page 415). It can consist of up to 31 characters and must be enclosed in single quotation marks. Do not specify this keyword if the user does not need a password. You can change the password every time you create a Symphony file (when creating a CP extension).

Attention: The password is *not encrypted.* You must take the necessary action to protect the password from unauthorized access.

One way to do this is to place the USRREC definitions in a separate member in a separate library. This library should then be protected with RACF so it can be accessed only by authorized persons. The library should be added in the EQQPARM data set concatenation in the end-to-end server started task and in the plan extend, replan, and Symphony renew batch jobs.

Example JCL for plan replan, extend, and Symphony renew batch jobs:

```
//EQQPARM  DD DISP=SHR,DSN=TWS.V8R20.PARMLIB(BATCHOPT)
//         DD DISP=SHR,DSN=TWS.V8R20.PARMUSR
```

In this example, the USRREC member is placed in the TWS.V8R20.PARMUSR library. This library can then be protected with RACF according to your standards. All other BATCHOPT initialization statements are placed in the usual parameter library. In the example, this library is named TWS.V8R20.PARMLIB and the member is BATCHOPT.

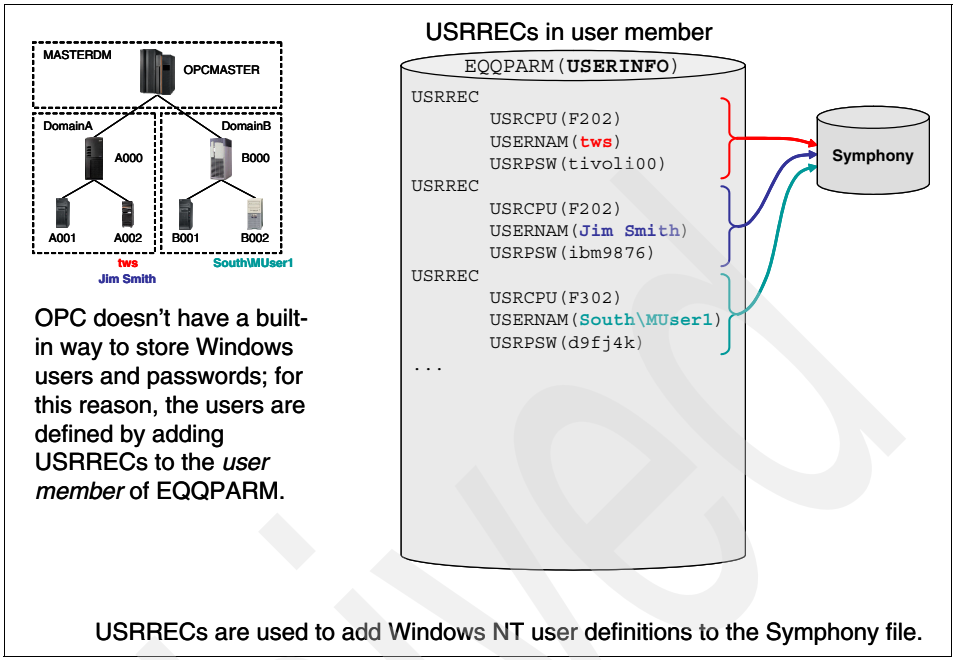


Figure 15-15 Example of three USRREC definitions: for a local and domain Windows user

15.1.8 Example of DOMREC and CPUREC definitions

We have explained how to use DOMREC and CPUREC statements to define the network topology for a Tivoli Workload Scheduler network in a Tivoli Workload Scheduler for z/OS end-to-end environment. We now use these statements to define a simple Tivoli Workload Scheduler network in Tivoli Workload Scheduler for z/OS.

As an example, Figure 15-16 on page 416 illustrates a simple Tivoli Workload Scheduler network. In this network there is one domain, DOMAIN1, under the master domain (MASTERDM).

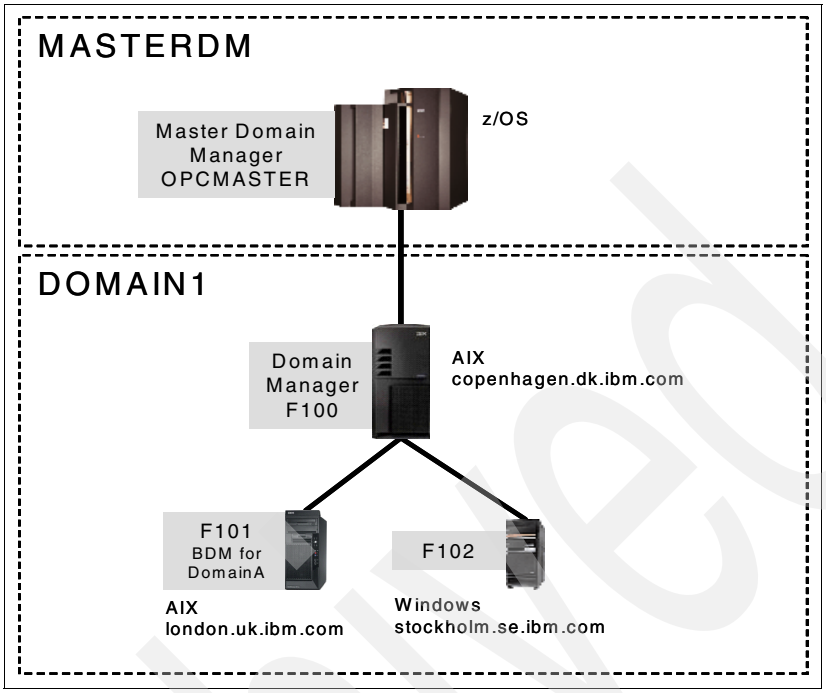


Figure 15-16 Simple end-to-end scheduling environment

Example 15-3 describes the DOMAIN1 domain with the DOMAIN topology statement.

Example 15-3 Domain definition

```
DOMREC    DOMAIN(DOMAIN1)              /* Name of the domain is DOMAIN1 */
          DOMMMNGR(F100)               /* F100 workst. is domain mng.   */
          DOMPARENT(MASTERDM)          /* Domain parent is MASTERDM     */
```

In end-to-end, the master domain (MASTERDM) is always the Tivoli Workload Scheduler for z/OS controller. (It is predefined and cannot be changed.) Because the DOMAIN1 domain is under the MASTERDM domain, MASTERDM must be defined in the DOMPARENT parameter. The DOM;MNGR keyword represents the name of the workstation.

There are three workstations (CPUs) in the DOMAIN1 domain. To define these workstations in the Tivoli Workload Scheduler for z/OS end-to-end network, we must define three CPURECs, one for each workstation (server) in the network.

Example 15-4 Workstation (CPUREC) definitions for the three FTWs

```
CPUREC   CPUNAME(F100)                /* Domain manager for DM100      */
         CPUOS(AIX)                   /* AIX operating system          */
         CPUNODE(copenhagen.dk.ibm.com)  /* IP address of CPU (DNS)    */
         CPUTCPIP(31281)              /* TCP port number of NETMAN     */
         CPUDOMAIN(DM100)             /* The TWS domain name for CPU   */
         CPUTYPE(FTA)                 /* This is a FTA CPU type        */
         CPUAUTOLNK(ON)               /* Autolink is on for this CPU   */
         CPUFULLSTAT(ON)              /* Full status on for DM         */
         CPURESDEP(ON)                /* Resolve dependencies on for DM*/
         CPULIMIT(20)                 /* Number of jobs in parallel    */
         CPUTZ(Europe/Copenhagen)     /* Time zone for this CPU        */
         CPUUSER(twstest)             /* default user for CPU          */
         SSLLEVEL(OFF)                /* SSL is not active             */
         SSLPORT(31113)               /* Default SSL port              */
         FIREWALL(NO)                 /* WS not behind firewall        */
CPUREC   CPUNAME(F101)                /* fault tolerant agent in DM100 */
         CPUOS(AIX)                   /* AIX operating system          */
         CPUNODE(london.uk.ibm.com)      /* IP address of CPU (DNS)    */
         CPUTCPIP(31281)              /* TCP port number of NETMAN     */
         CPUDOMAIN(DM100)             /* The TWS domain name for CPU   */
         CPUTYPE(FTA)                 /* This is a FTA CPU type        */
         CPUAUTOLNK(ON)               /* Autolink is on for this CPU   */
         CPUFULLSTAT(ON)              /* Full status on for BDM        */
         CPURESDEP(ON)                /* Resolve dependencies on BDM   */
         CPULIMIT(20)                 /* Number of jobs in parallel    */
         CPUSERVER(A)                 /* Start extra mailman process   */
         CPUTZ(Europe/London)         /* Time zone for this CPU        */
         CPUUSER(maestro)             /* default user for ws           */
         SSLLEVEL(OFF)                /* SSL is not active             */
         SSLPORT(31113)               /* Default SSL port              */
         FIREWALL(NO)                 /* WS not behind firewall        */
CPUREC   CPUNAME(F102)                /* fault tolerant agent in DM100 */
         CPUOS(WNT)                   /* Windows operating system      */
         CPUNODE(stockholm.se.ibm.com)   /* IP address for CPU (DNS)   */
         CPUTCPIP(31281)              /* TCP port number of NETMAN     */
         CPUDOMAIN(DM100)             /* The TWS domain name for CPU   */
         CPUTYPE(FTA)                 /* This is a FTA CPU type        */
         CPUAUTOLNK(ON)               /* Autolink is on for this CPU   */
         CPUFULLSTAT(OFF)             /* Full status off for FTA       */
         CPURESDEP(OFF)               /* Resolve dependencies off FTA  */
         CPULIMIT(10)                 /* Number of jobs in parallel    */
         CPUSERVER(A)                 /* Start extra mailman process   */
         CPUTZ(Europe/Stockholm)      /* Time zone for this CPU        */
```

```
CPUUSER(twstest)            /* default user for ws          */
SSLLEVEL(OFF)               /* SSL is not active            */
SSLPORT(31113)              /* Default SSL port             */
FIREWALL(NO)                /* WS not behind firewall       */
```

Because F101 will be backup domain manager for F100, F101 is defined with CPUFULLSTATUS (ON) and CPURESDEP(ON).

F102 is a fault-tolerant agent without extra responsibilities, so it is defined with CPUFULLSTATUS(OFF) and CPURESDEP(OFF) because dependency resolution within the domain is the task of the domain manager. This improves performance by reducing network traffic.

Note: CPUOS(WNT) applies for all Windows platforms.

Finally, because F102 runs on a Windows server™, we must create at least one USRREC definition for this server. In our example, we would like to be able to run jobs on the Windows server under either the Tivoli Workload Scheduler installation user (twstest) or the database user, databusr.

Example 15-5 USRREC definition for tws F102 Windows users, twstest and databusr

```
USRREC   USRCPU(F102)             /* Definition for F102 Windows CPU */
         USRNAM(twstest)          /* The user name (local user)      */
         USRPSW('twspw01')        /* The password for twstest        */
USRREC   USRCPU(F102)             /* Definition for F102 Windows CPU */
         USRNAM(databusr)         /* The user name (local user)      */
         USRPSW('data01ad')       /* Password for databusr           */
```

15.1.9 The JTOPTS TWSJOBNAME() parameter

With the JTOPTS TWSJOBNAME() parameter, it is possible to specify different criteria that Tivoli Workload Scheduler for z/OS should use when creating the job name in the Symphony file in USS.

The syntax for the JTOPTS TWSJOBNAME() parameter is:

```
TWSJOBNAME(EXTNAME|EXTNOCC|JOBNAME|OCCNAME)
```

If you do not specify the TWSJOBNAME() parameter, the value OCCNAME is used by default.

When choosing OCCNAME, the job names in the Symphony file will be generated with one of the following formats:

- <X>_<Num>_<Application Name> when the job is created in the Symphony file
- <X>_<Num>_<Ext>_<Application Name> when the job is first deleted and then re-created in the current plan

  In these examples, <X> can be J for normal jobs (operations), P for jobs representing pending predecessors, and R for recovery jobs.

  <Num> is the operation number.

  <Ext> is a sequential decimal number that is increased every time an operation is deleted and then re-created.

  <Application Name> is the name of the occurrence the operation belongs to.

Figure 15-17 on page 420 shows an example of how the job names (and job stream names) are generated by default in the Symphony file when JTOPTS TWSJOBNAME(OCCNAME) is specified or defaulted.

Note that *occurrence* in Tivoli Workload Scheduler for z/OS is the same as JSC job stream instance (that is, a job stream or an application that is on the plan in Tivoli Workload Scheduler for z/OS).

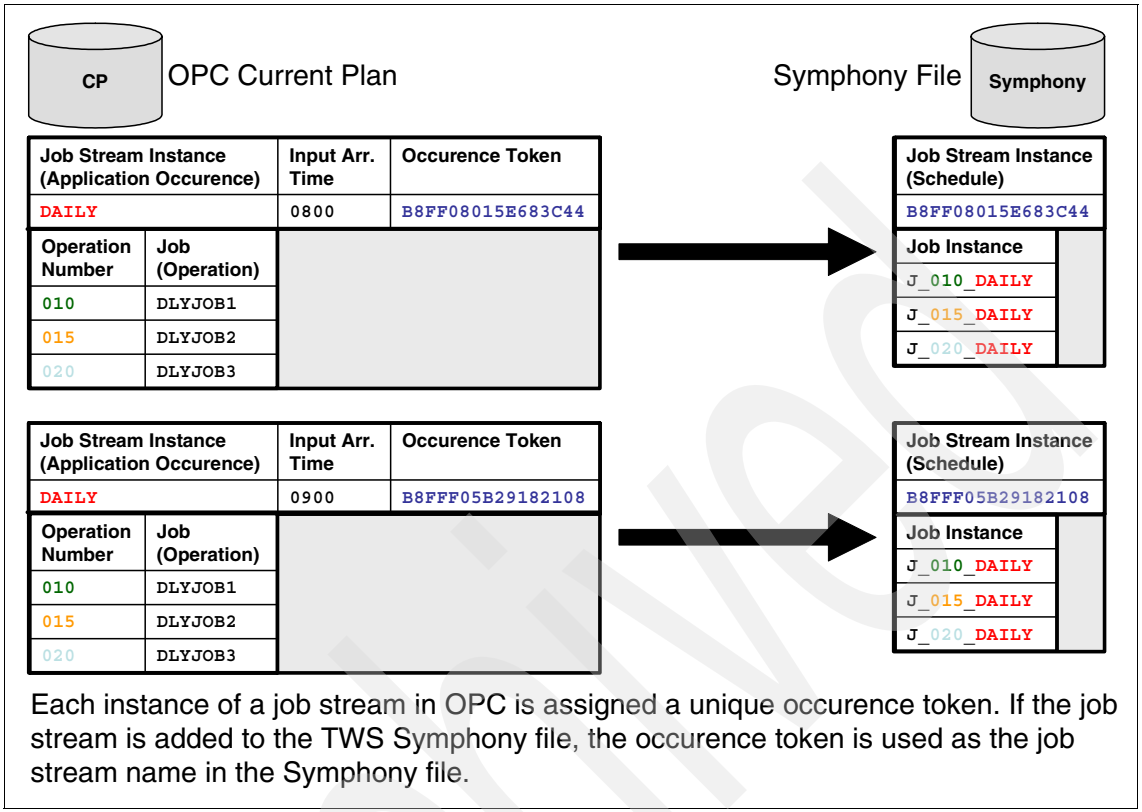


Figure 15-17 Generation of job and job stream names in the Symphony file

If any of the other values (EXTNAME, EXTNOCC, or JOBNAME) is specified in the JTOPTS TWSJOBNAME() parameter, the job name in the Symphony file is created according to one of the following formats:

- <X><Num>_<JobInfo> when the job is created in the Symphony file
- <X><Num>_<Ext>_<JobInfo> when the job is first deleted and then re-created in the current plan

In these examples:

<*X*> can be J for normal jobs (operations), P for jobs representing pending predecessors, and R for recovery jobs. For jobs representing pending predecessors, the job name is in all cases generated by using the OCCNAME criterion. This is because, in the case of pending predecessors, the current plan does not contain the required information (excepting the name of the occurrence) to build the Symphony name according to the other criteria.

<Num> is the operation number.

<Ext> is the hexadecimal value of a sequential number that is increased every time an operation is deleted and then re-created.

<JobInfo> depends on the chosen criterion:

- For EXTNAME: <JobInfo> is filled with the first 32 characters of the extended job name associated with that job (if it exists) or with the eight-character job name (if the extended name does not exist). Note that the extended job name, in addition to being defined in the database, must also exist in the current plan.
- For EXTNOCC: <JobInfo> is filled with the first 32 characters of the extended job name associated with that job (if it exists) or with the application name (if the extended name does not exist). Note that the extended job name, in addition to being defined in the database, must also exist in the current plan.
- For JOBNAME: <JobInfo> is filled with the eight-character job name.

The criterion that is used to generate a Tivoli Workload Scheduler job name will be maintained throughout the entire life of the job.

Note: In order to choose the EXTNAME, EXTNOCC, or JOBNAME criterion, the EQQTWSOU data set must have a record length of 160 bytes. Before using any of the above keywords, you *must* migrate the EQQTWSOU data set if you have allocated the data set with a record length less than 160 bytes. Sample EQQMTWSO is available to migrate this data set from record length 120 to 160 bytes.

Limitations when using the EXTNAME and EXTNOCC criteria:

- The job name in the Symphony file can contain only alphanumeric characters, dashes, and underscores. All other characters that are accepted for the extended job name are converted into dashes. Note that a similar limitation applies with JOBNAME: When defining members of partitioned data sets (such as the script or the job libraries), national characters can be used, but they are converted into dashes in the Symphony file.
- The job name in the Symphony file must be in uppercase. All lowercase characters in the extended name are automatically converted to uppercase by Tivoli Workload Scheduler for z/OS.

Note: Using the job name (or the extended name as part of the job name) in the Symphony file implies that it becomes a key for identifying the job. This also means that the extended name - job name is used as a key for addressing all events that are directed to the agents. For this reason, be aware of the following facts for the operations that are included in the Symphony file:

- Editing the extended name is inhibited for operations that are created when the TWSJOBNAME keyword was set to EXTNAME or EXTNOCC.
- Editing the job name is inhibited for operations created when the TWSJOBNAME keyword was set to EXTNAME or JOBNAME.

15.1.10 Verify end-to-end installation

When all installation tasks as described in the previous sections have been completed, and all initialization statements and data sets related to end-to-end scheduling have been defined in the Tivoli Workload Scheduler for z/OS controller, end-to-end server, and plan extend, replan, and Symphony renew batch jobs, it is time to do the first verification of the mainframe part.

Note: This verification can be postponed until workstations for the fault-tolerant agents have been defined in Tivoli Workload Scheduler for z/OS and, optionally, Tivoli Workload Scheduler has been installed on the fault-tolerant agents (the Tivoli Workload Scheduler servers or agents).

Verify the Tivoli Workload Scheduler for z/OS controller

After the customization steps have been completed, simply start the Tivoli Workload Scheduler controller. Check the controller message log (EQQMLOG) for any unexpected error or warning messages. All Tivoli Workload Scheduler z/OS messages are prefixed with EQQ. See *IBM Tivoli Workload Scheduler for z/OS Messages and Codes* V8.2 (Maintenance Release April 2004), SC32-1267.

Because we have activated the end-to-end feature in the controller initialization statements by specifying the OPCOPTS TPLGYSRV() parameter and we have asked the controller to start our end-to-end server by the SERVERS(TWSCE2E) parameter, we will see messages as shown in Example 15-6 in the Tivoli Workload Scheduler for z/OS controller message log (EQQMLOG).

Example 15-6 Tivoli Workload Scheduler for z/OS controller messages for end-to-end

```
EQQZ005I OPC SUBTASK E2E ENABLER      IS BEING STARTED
EQQZ085I OPC SUBTASK E2E SENDER       IS BEING STARTED
EQQZ085I OPC SUBTASK E2E RECEIVER     IS BEING STARTED
EQQG001I SUBTASK E2E ENABLER HAS STARTED
```

```
EQQG001I SUBTASK E2E SENDER HAS STARTED
EQQG001I SUBTASK E2E RECEIVER HAS STARTED
EQQW097I END-TO-END RECEIVER STARTED SYNCHRONIZATION WITH THE EVENT
MANAGER
EQQW097I      0 EVENTS IN EQQTWSIN WILL BE REPROCESSED
EQQW098I END-TO-END RECEIVER FINISHED SYNCHRONIZATION WITH THE EVENT
MANAGER
EQQ3120E END-TO-END TRANSLATOR SERVER PROCESS IS NOT AVAILABLE
EQQZ193I END-TO-END TRANSLATOR SERVER PROCESSS NOW IS AVAILABLE
```

Note: If you do not see all of these messages in your controller message log, you probably have not applied all available service updates.

The messages in the previous example are extracted from the Tivoli Workload Scheduler for z/OS controller message log. You will see several other messages between those messages if you look in your controller message log.

If the Tivoli Workload Scheduler for z/OS controller is started with empty EQQTWSIN and EQQTWSOU data sets, messages shown in Example 15-7 will be issued in the controller message log (EQQMLOG).

Example 15-7 Formatting messages when EQQTWSOU and EQQTWSIN are empty

```
EQQW030I A DISK DATA SET WILL BE FORMATTED, DDNAME = EQQTWSOU
EQQW030I A DISK DATA SET WILL BE FORMATTED, DDNAME = EQQTWSIN
EQQW038I A DISK DATA SET HAS BEEN FORMATTED, DDNAME = EQQTWSOU
EQQW038I A DISK DATA SET HAS BEEN FORMATTED, DDNAME = EQQTWSIN
```

Note: In the Tivoli Workload Scheduler for z/OS system messages, there will also be two IEC031I messages related to the formatting messages in Example 15-7. These messages can be ignored because they are related to the formatting of the EQQTWSIN and EQQTWSOU data sets.

The IEC031I messages look like:

```
IEC031I D37-04,IFG0554P,TWSC,TWSC,EQQTWSOU,.....................
IEC031I D37-04,IFG0554P,TWSC,TWSC,EQQTWSIN,.................
```

The messages in the next two examples show that the controller is started with the end-to-end feature active and that it is ready to run jobs in the end-to-end environment.

When the Tivoli Workload Scheduler for z/OS controller is stopped, the end-to-end related messages shown in Example 15-8 will be issued.

Example 15-8 Controller messages for end-to-end when controller is stopped

```
EQQG003I SUBTASK E2E RECEIVER HAS ENDED
EQQG003I SUBTASK E2E SENDER HAS ENDED
EQQZ034I OPC SUBTASK E2E SENDER       HAS ENDED.
EQQZ034I OPC SUBTASK E2E RECEIVER     HAS ENDED.
EQQZ034I OPC SUBTASK E2E ENABLER      HAS ENDED.
```

Verify the Tivoli Workload Scheduler for z/OS server

After the customization steps have been completed for the Tivoli Workload Scheduler end-to-end server started task, simply start the end-to-end server started task. Check the server message log (EQQMLOG) for any unexpected error or warning messages. All Tivoli Workload Scheduler z/OS messages are prefixed with EQQ. See the *IBM Tivoli Workload Scheduler for z/OS Messages and Codes, Version 8.2 (Maintenance Release April 2004),* SC32-1267.

When the end-to-end server is started for the first time, check that the messages shown in Example 15-9 appear in the Tivoli Workload Scheduler for z/OS end-to-end server EQQMLOG.

Example 15-9 End-to-end server messages the first time the end-to-end server is started

```
EQQPH00I SERVER TASK HAS STARTED
EQQPH33I THE END-TO-END PROCESSES HAVE BEEN STARTED
EQQZ024I Initializing wait parameters
EQQPT01I Program "/usr/lpp/TWS/TWS810/bin/translator" has been started,
         pid is 67371783
EQQPT01I Program "/usr/lpp/TWS/TWS810/bin/netman" has been started,
         pid is 67371919
EQQPT56W The /DD:EQQTWSIN queue has not been formatted yet
EQQPT22I Input Translator thread stopped until new Symphony will be available
```

The messages shown in Example 15-9 are normal when the Tivoli Workload Scheduler for z/OS end-to-end server is started for the first time and there is no Symphony file created.

Furthermore, the end-to-end server message EQQPT56W normally is issued only for the EQQTWSIN data set, if the EQQTWSIN and EQQTWSOU data sets are both empty and there is no Symphony file created.

If the Tivoli Workload Scheduler for z/OS controller and end-to-end server is started with an empty EQQTWSOU data set (for example, reallocated with a new record length), message EQQPT56W will be issued for the EQQTWSOU data set:

```
EQQPT56W The /DD:EQQTWSOU queue has not been formatted yet
```

If a Symphony file has been created, the end-to-end server messages log contains the messages in Example 15-10.

Example 15-10 End-to-end server messages when server is started with Symphony file

```
EQQPH33I THE END-TO-END PROCESSES HAVE BEEN STARTED
EQQZ024I Initializing wait parameters
EQQPT01I Program "/usr/lpp/TWS/TWS820/bin/translator" has been started,
         pid is 33817341
EQQPT01I Program "/usr/lpp/TWS/TWS820/bin/netman" has been started,
         pid is 262958
EQQPT20I Input Translator waiting for Batchman and Mailman are started
EQQPT21I Input Translator finished waiting for Batchman and Mailman
```

The messages shown in Example 15-10 are the normal start-up messages for a Tivoli Workload Scheduler for z/OS end-to-end server with a Symphony file.

When the end-to-end server is stopped, the messages shown in Example 15-11 should be issued in the EQQMLOG.

Example 15-11 End-to-end server messages when server is stopped

```
EQQZ000I A STOP OPC COMMAND HAS BEEN RECEIVED
EQQPT04I Starter has detected a stop command
EQQPT40I Input Translator thread is shutting down
EQQPT12I The Netman process (pid=262958) ended successfully
EQQPT40I Output Translator thread is shutting down
EQQPT53I Output Translator thread has terminated
EQQPT53I Input Translator thread has terminated
EQQPT40I Input Writer thread is shutting down
EQQPT53I Input Writer thread has terminated
EQQPT12I The Translator process (pid=33817341) ended successfully
EQQPT10I All Starter's sons ended
EQQPH34I THE END-TO-END PROCESSES HAVE ENDED
EQQPH01I SERVER TASK ENDED
```

After successful completion of the verification, move on to the next step in the end-to-end installation.

15.2 Installing FTAs in an end-to-end environment

In this section, we describe how to install Tivoli Workload Scheduler FTAs (also referred as *fault-tolerant workstations* in end-to-end scheduling), in an end-to-end environment.

Important: Maintenance releases of Tivoli Workload Scheduler are made available about every three months. We recommend that, before installing, check for the latest available update at:

`ftp://ftp.software.ibm.com`

Installing a Tivoli Workload Scheduler agent in an end-to-end environment is not very different from installing Tivoli Workload Scheduler when Tivoli Workload Scheduler for z/OS is not involved. Follow the installation instructions in the *IBM Tivoli Workload Scheduler Planning and Installation Guide,* SC32-1273. The main differences to keep in mind are that in an end-to-end environment, the master domain manager is always the Tivoli Workload Scheduler for z/OS engine (known by the Tivoli Workload Scheduler workstation name OPCMASTER), and the local workstation name of the fault-tolerant workstation is limited to four characters.

Important: The /usr/unison/components file is used *only* on Tier 2 platforms.

Important: Do not edit or remove the /etc/TWS/TWS Registry.dat file because this could cause problems with uninstalling Tivoli Workload Scheduler or with installing fix packs. Do not remove this file unless you intend to remove all installed Tivoli Workload Scheduler V8.2 engines from the computer.

Certain prerequisites must be met before you run the installation program:

- This particular method of installation uses a Java Virtual Machine, and thus requires specific system requirements. The supported operating systems for the ISMP and Silent Install are: Red Hat Linux for Intel®; Red Hat Linux for S/390®; Sun™ Solaris; HP-UX; AIX; Windows NT; Windows 2000 and 2003 Professional, Server and Advanced Server; and Windows XP Professional.
- On UNIX workstations only, you must create the user login account for which you are installing the product before running the installation, if it does not already exist. You must make sure that your UNIX system is not configured to require a password when the **su** command is issued by root, otherwise the installation will fail.
- On Windows systems, your login account must be a member of the Local Windows Administrators group. You must have full privileges for administering the system. However, if your installation includes the Tivoli Management Framework, then you must be logged on as the local Administrator (not a domain Administrator) on the workstation on which you are installing. Note that the local and domain Administrator user names are case sensitive, so check the Users and Passwords panel for the correct case. you must log on to

the workstation on which you are installing with the correct spelling and case or the installation will fail.

- If your installation will include the Tivoli Management Framework, you need access to the images of the Tivoli Management Framework and the Tivoli Management Framework language packs.
- If you will access installation images from a mapped drive, the drive must be mapped by the user who is logged on to the system performing the installation.
- Only one ISMP installation session at a time can run on the same workstation.

15.2.1 Installation program and CDs

When you install IBM Tivoli Workload Scheduler using the installation program, the registry file is checked to determine whether other IBM Tivoli Workload Scheduler V 8.2 instances are already installed. Now multiple copies of the product can be installed on a single computer if a unique name and installation path are used for each instance.

So on Tier 1 platforms, when you install IBM Tivoli Workload Scheduler using the ISMP installation program or the twsinst script, a check is performed to determine whether there are other instances installed as indicated in the prior paragraph. The TWSRegistry.dat file stores the history of all instances installed, and this is the sole purpose of this file. The presence of this file is not essential for the functioning of the product. On Windows platforms, this file is stored under the system drive directory (for example, c:\winnt\system32). On UNIX platforms, this file is stored in the /etc/TWS path. The file contains values of the attributes that define an IBM Tivoli Workload Scheduler installation (Table 15-4).

Table 15-4 TWSRegistry.dat file attributes

Attribute	Value
ProductID	TWS_ENGINE
PackageName	Name of the software package used to perform the installation
InstallationPath	Absolute path of the IBM Tivoli Workload Scheduler instance
UserOwner	The owner of the installation
MajorVersion	IBM Tivoli Workload Scheduler release number
MinorVersion	IBM Tivoli Workload Scheduler version number
MaintenanceVersion	IBM Tivoli Workload Scheduler maintenance version number
PatchVersion	The latest product patch number installed

Attribute	Value
Agent	Any one of the following: standard agent, fault-tolerant agent, master domain manager
FeatureList	The list of optional features installed
LPName	The name of the software package block that installs the language pack
LPList	A list of all languages installed for the instance installed

Example 15-12 shows the TWSRegistry.dat file on a master domain manager.

Example 15-12 IBM Tivoli Workload Scheduler TWSRegistry.dat file

```
/Tivoli/Workload_Scheduler/tws_nord_DN_objectClass=OU
/Tivoli/Workload_Scheduler/tws_nord_DN_PackageName=TWS_NT_tws_nord.8.2
/Tivoli/Workload_Scheduler/tws_nord_DN_MajorVersion=8
/Tivoli/Workload_Scheduler/tws_nord_DN_MinorVersion=2
/Tivoli/Workload_Scheduler/tws_nord_DN_PatchVersion=
/Tivoli/Workload_Scheduler/tws_nord_DN_FeatureList=TBSM
/Tivoli/Workload_Scheduler/tws_nord_DN_ProductID=TWS_ENGINE
/Tivoli/Workload_Scheduler/tws_nord_DN_ou=tws_nord
/Tivoli/Workload_Scheduler/tws_nord_DN_InstallationPath=c:\TWS\tws_nord
/Tivoli/Workload_Scheduler/tws_nord_DN_UserOwner=tws_nord
/Tivoli/Workload_Scheduler/tws_nord_DN_MaintenanceVersion=
/Tivoli/Workload_Scheduler/tws_nord_DN_Agent=MDM
```

For product installations on Tier 2 platforms, product groups are defined in the components file. This file permits multiple copies of a product to be installed on a single computer by designating a different user for each copy. If the file does not exist prior to installation, it is created by the customize script, as shown in the following sample.

Product	Version	Home directory	Product group
maestro	8.2	/data/maestro8/maestro	TWS_maestro8_8.2

Entries in the file are automatically made and updated by the customize script.

On UNIX, the file name of the components file is defined in the variable UNISON_COMPONENT_FILE.

If the variable is not set, customize uses the file name /usr/unison/components.

After the installation or an upgrade, you will be able to view the contents of the components file on a Tier 2 platform by running the **ucomp** program as follows:

```
ucomp -l
```

These CDs are required to start the installation process:

- *IBM Tivoli Workload Scheduler Installation Disk 1*: Includes images for AIX, Solaris, HP-UX, and Windows
- *IBM Tivoli Workload Scheduler Installation Disk 2*: Includes images for Linux and Tier 2 platforms.

For Windows, the SETUP.EXE file is located in the Windows folder on Disk 1.

On UNIX platforms, there are two different SETUP.bin files. The first is located in the root directory of the installation CD, and the second is located in the folder of the UNIX operating system on which you are installing.

1. At the start of the installation process (whether on Windows, AIX, or whatever machine you are perfroming the installation on), the GUI option first lets you select the language you wish to use when doing the installation (Figure 15-18). From the pull-down menu, you can select additional languages such as: French, German, Italian, Japanese, Korean, Portuguese (Brazil), Simplified Chinese, Spanish, and Traditional Chinese.

Figure 15-18 Language Selection Menu

2. After selecting your language, you will see the IBM Tivoli Workload Scheduler Installation window shown in Figure 15-19.

   The installation offers three operations:

   – A fresh install of IBM Tivoli Workload Scheduler
   – Adding functionality or modifying your existing IBM Tivoli Workload Scheduler installation
   – Upgrading from a previous version

   Click **Next**.

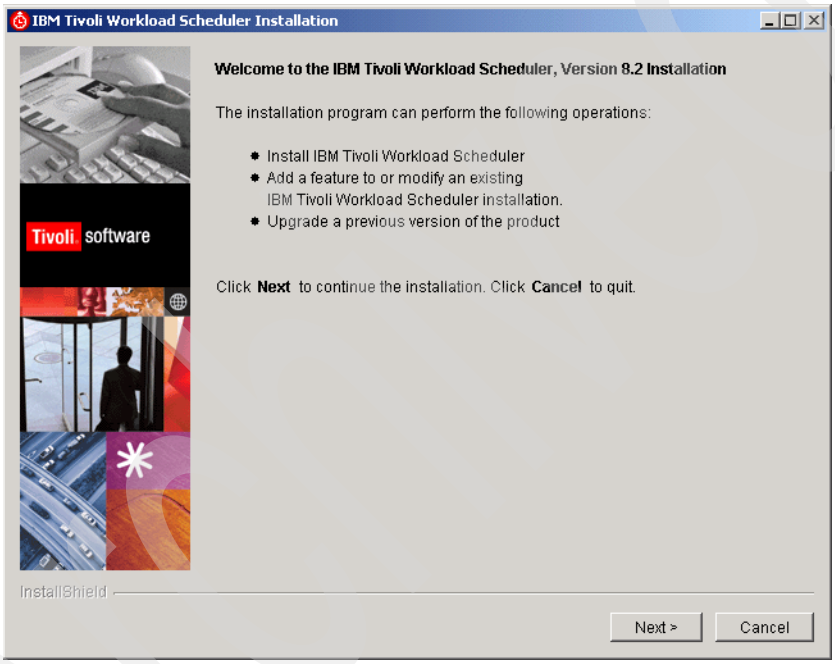


Figure 15-19 IBM Tivoli Workload Scheduler Installation window

3. Accept the IBM Tivoli Workload Scheduler License agreement (Figure 15-20).

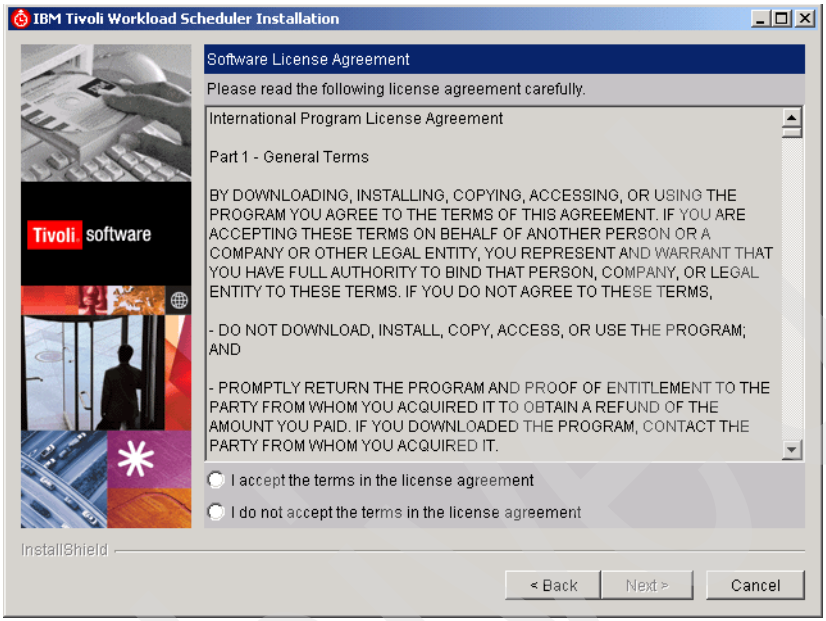


Figure 15-20 IBM Tivoli Workload Scheduler License Agreement

4. The Installation window opens. Figure 15-21 shows that the product has determined that this is a new installation of IBM Tivoli Workload Scheduler.

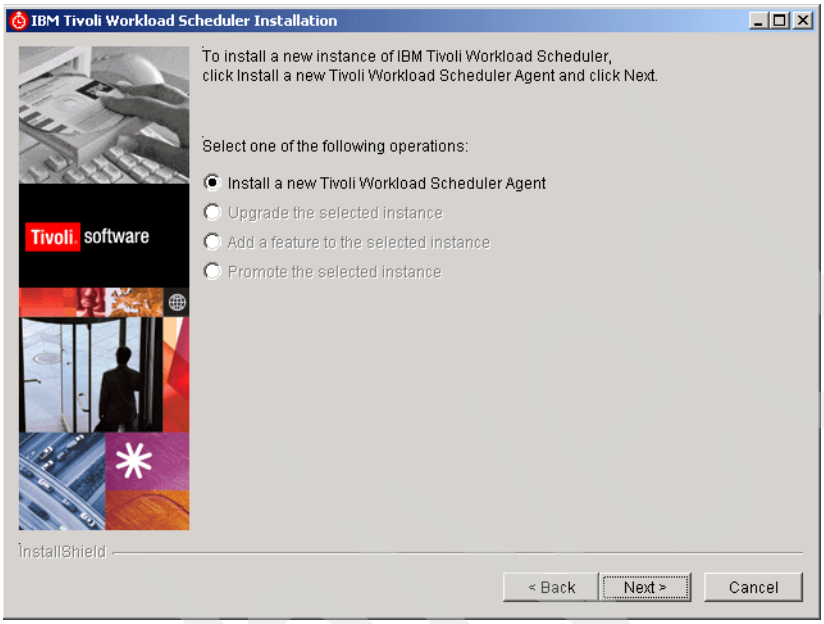


Figure 15-21 Install a new Tivoli Workload Scheduler Agent

5. Designate your user name and password screen; spaces are not allowed in either (Figure 15-22). On Windows systems, if this user account does not already exist, it is automatically created by the installation program. If you specify a domain user, specify the name as *domain_name\user_name*. If you specify a local user with the same name as a domain user, the local user must first be created manually by an Administrator and then specified as *system_name\user_name*.

   Type and confirm the password, which must comply with the password policy in your Local Security Settings; otherwise, the installation will fail.

   Note: On UNIX systems, this user account must be created manually before running the installation program. Create a user with a home directory. IBM Tivoli Workload Scheduler will be installed under the HOME directory of the selected user.

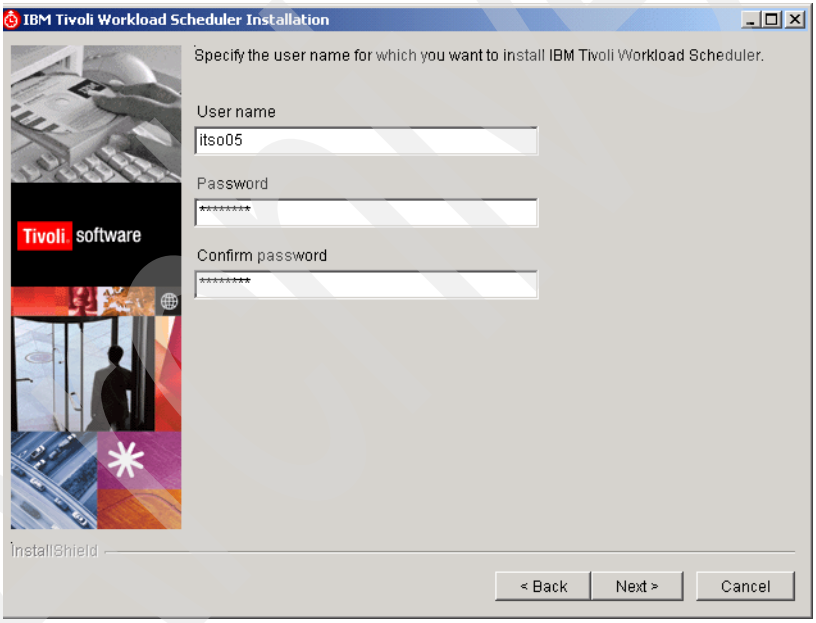


Figure 15-22 User name and password window

6. Because this is a new installation, the window in Figure 15-23 appears, specifying that the user that you just created does not exist and will be created with the rights shown.

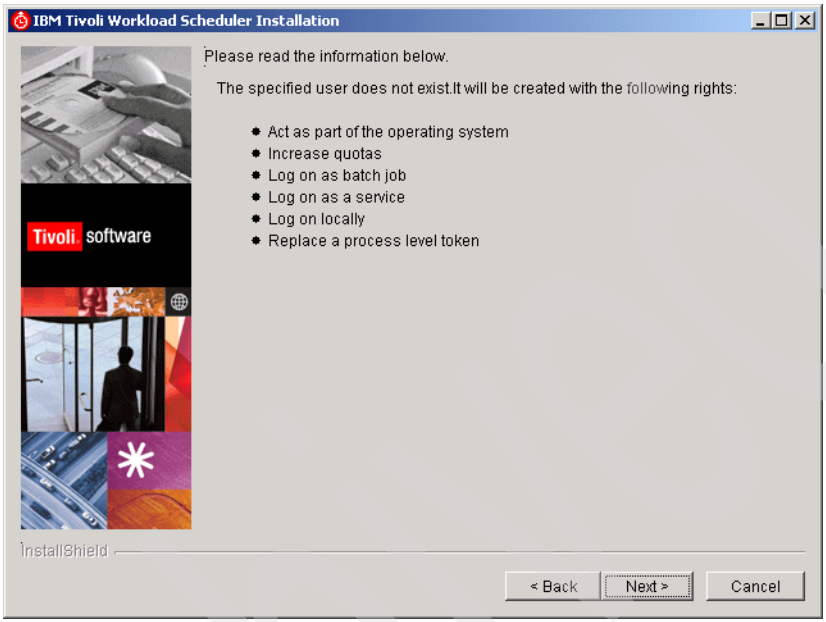


Figure 15-23 IBM Tivoli Workload Scheduler Installation new user

7. Designate the directory where you want to install IBM Tivoli Workload Scheduler (Figure 15-24). If you create a new directory, its name cannot contain spaces. For Windows systems, this directory must be located on an NTFS file system.

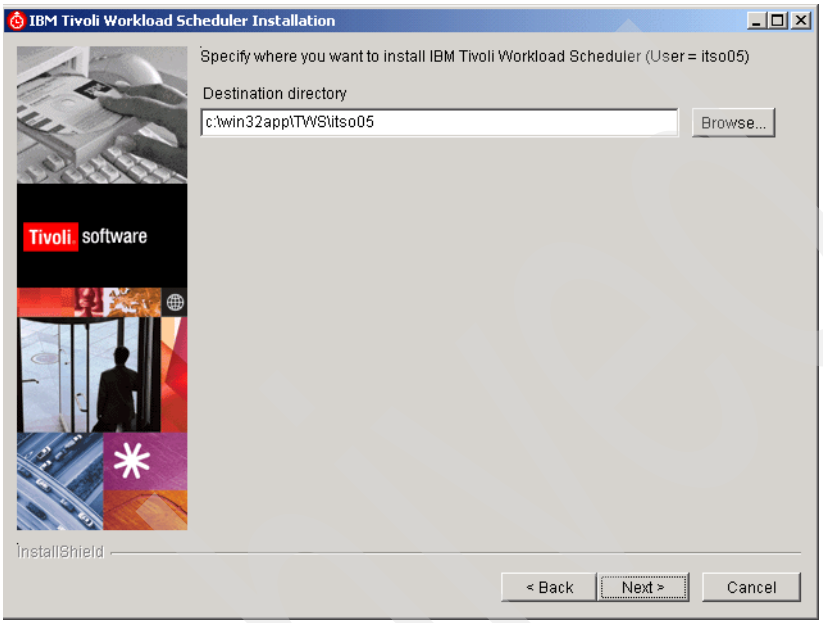


Figure 15-24 IBM Tivoli Workload Scheduler Installation Directory

8. Choose the type of installation (Figure 15-25):
   - **Typical** installs a fault-tolerant agent based on the language you selected previously.
   - **Custom** enables you to select the type of agent you want to install.
   - **Full** installs a master domain manager as well as the IBM Tivoli Workload Scheduler Connector and its prerequisites, which includes Tivoli Management Framework and the Tivoli Job Scheduling Services.

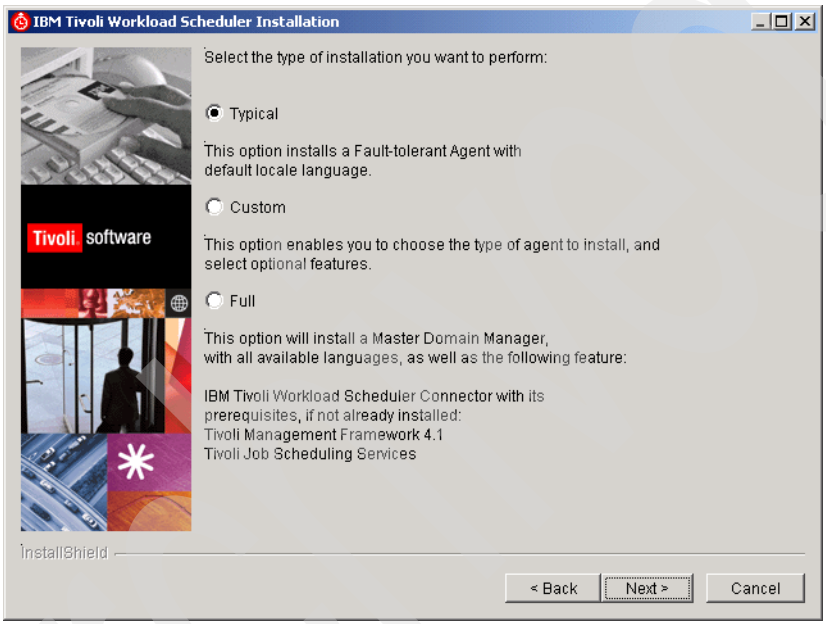


Figure 15-25 IBM Tivoli Workload Scheduler Installation type

Selecting either Typical (the default) or Full opens the window for specifying the workstation configuration information for the agent (Figure 15-26).

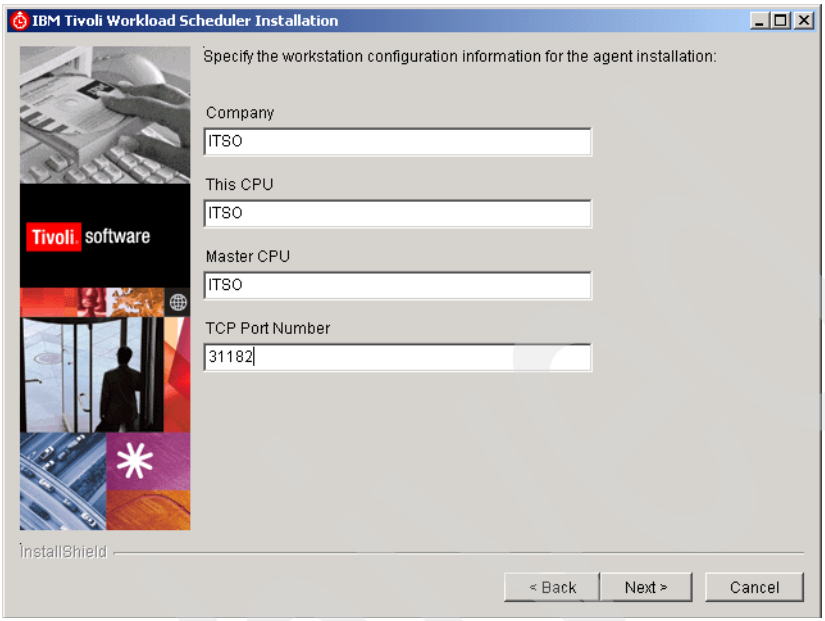


Figure 15-26 IBM Tivoli Workload Scheduler workstation configuration

Table 15-5 explains the fields in this window.

Table 15-5 Explanation of the fields for the workstation configuration window

Field	Value
Company	Type the company name. This name appears in program headers and reports. Spaces are permitted, provided that the name is not enclosed in double quotation marks.
This CPU	Type the IBM Tivoli Workload Scheduler name of the workstation. This name cannot exceed 16 characters and cannot contain spaces.
Master CPU	Type the name of the master domain manager. This name cannot exceed 16 characters and cannot contain spaces.
TCP Port Number	The TCP port number used by the instance being installed. It must be a value in the range 1 – 65535. The default is 31111. When installing more than one instance on the same workstation, use different port numbers for each instance.

If you choose Custom, you have the choice of the type of agent you want to install: Standard, Fault Tolerant (same for Extended Agent), Backup, or Master Domain Manager (Figure 15-27). Making a selection and clicking Next opens the window in Figure 15-26 on page 437.

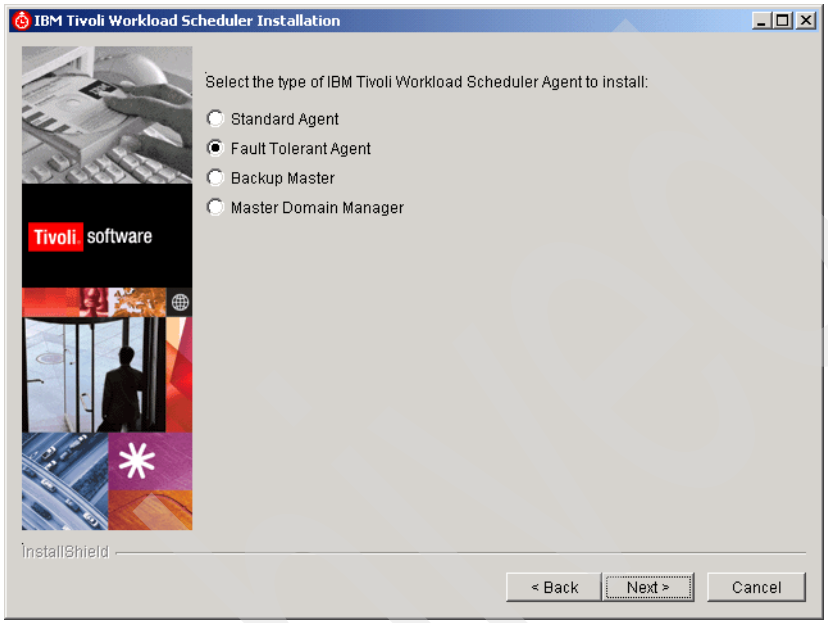


Figure 15-27 IBM Tivoli Workload Scheduler Custom Installation options

9. Designate the connector name to be associated with the agent installation (Figure 15-28 on page 439). This name will be displayed in the Job Scheduling tree of the Job Scheduling Console (JSC). To avoid any confusion, use a name that includes the name of the fault-tolerant agent.

   If you plan to install the connector on several fault-tolerant agents in a network, keep in mind that the instance names must be unique both within the IBM Tivoli Workload Scheduler network and the Tivoli Management Region.

   The connector is an IBM Tivoli Management Framework service that enables the Job Scheduling Console clients to communicate with the IBM Tivoli Workload Scheduler engine. A connector can be installed on a system that must also be a Tivoli server or managed node.

   If you want to install the connector in your IBM Tivoli Workload Scheduler domain but you have no existing regions and you are not interested in implementing a full Tivoli management environment, then you should install the Tivoli Management Framework as a unique region (and therefore install as a Tivoli server) on each node that will run the connector.

You can even install connectors on workstations other than the master domain manager. This enables you to view the version of the Symphony file of this particular workstation. This may be important for using the Job Scheduling Console to manage the local parameters database or to submit command directly to the workstation rather than submitting through the master.

The workstation on which you install the connector must be either a managed node or a Tivoli server in the Tivoli Workload Scheduler database. You must install the connector on the master domain manager configured as a Tivoli server or managed node.

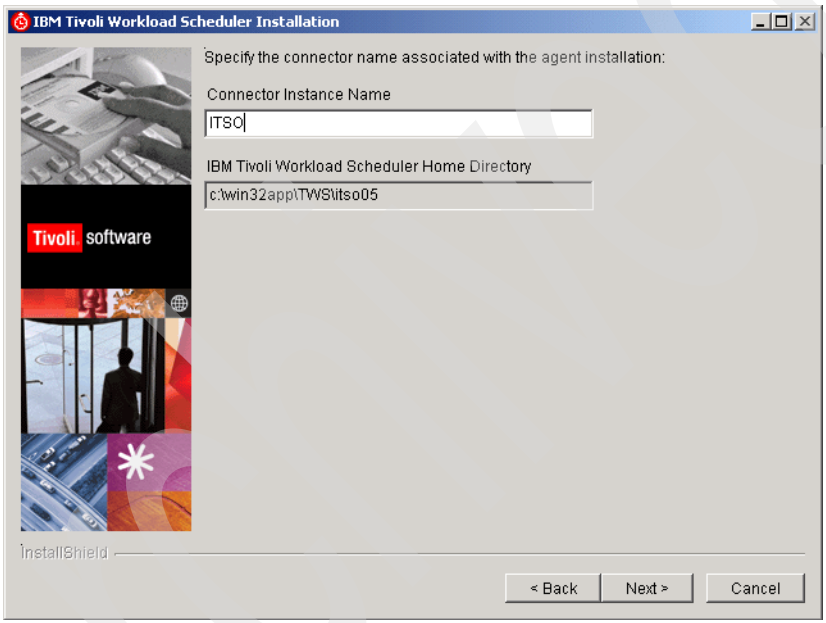


Figure 15-28 IBM Tivoli Workload Scheduler Connector

10. You have the option of installing additional languages (Figure 15-29). Choose any or all of the listed languages, or simply click **Next** to move on without adding any languages.

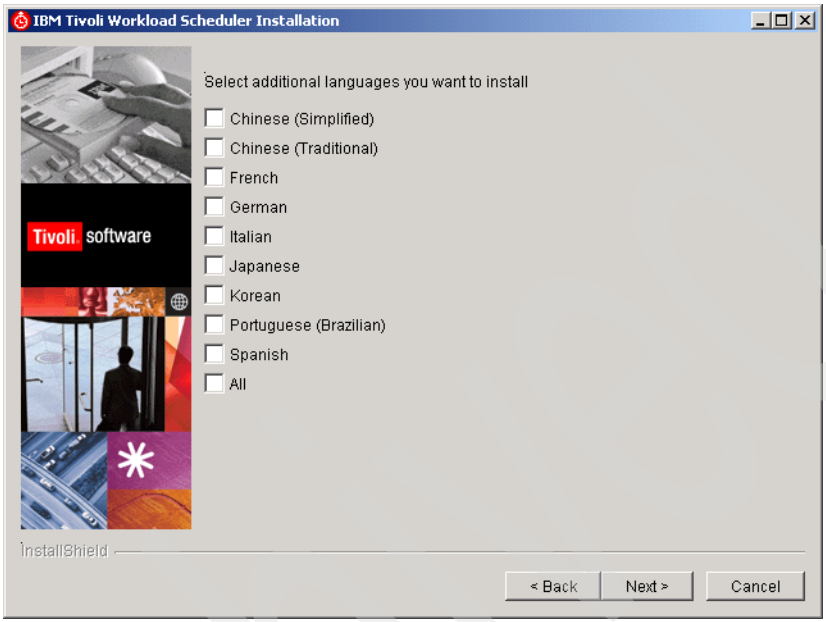


Figure 15-29 IBM Tivoli Workload Scheduler additional languages

11. Designate the location of the IBM Tivoli Workload Scheduler V 8.2 Tivoli Management Framework (Figure 15-30).

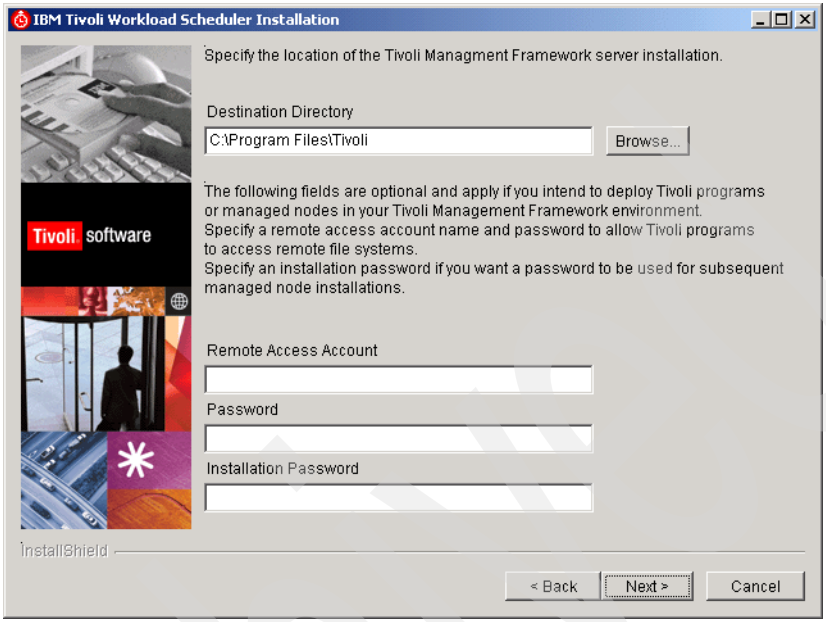


Figure 15-30 IBM Tivoli Workload Scheduler Tivoli Management Framework

12. The summary window in Figure 15-31 shows the directory where IBM Tivoli Workload Scheduler V8.2 will be installed and any additional features that will be added. Click **Next** to conclude the installation.

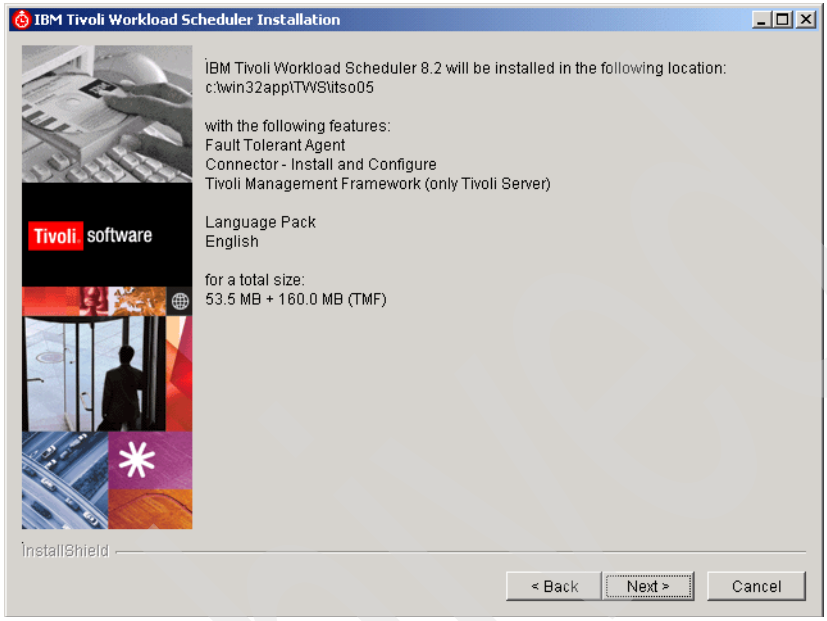


Figure 15-31 IBM Tivoli Workload Scheduler Installation location

15.2.2 Configuring steps for post-installation

After the installation of the FTWs, perform the additional configuration steps that are outlined in this section.

Configuring steps for Windows

On Windows systems, edit the PATH system variable to include TWShome and TWShome\bin.

For example, if IBM Tivoli Workload Scheduler has been installed in the c:\win32app\TWS\jdoe directory, the PATH variable should include this:

```
PATH=\win32app\TWS\jdoe;\win32app\TWS\jdoe\bin
```

Create the TWS_TISDIR environment variable and assign TWShome as the value. In this way, the necessary environment variables and search paths are set to enable you to run commands even if you are not located in the TWShome path. Alternatively, you can run the tws_env.cmd shell script to set up both the PATH and TWS_TISDIR variables.

Configuring steps for UNIX

For UNIX systems, create a .profile file for the TWSuser, if one does not already exist (TWShome/.profile). Edit the file and modify the PATH variable to include TWShome and TWShome/bin.

For example, if IBM Tivoli Workload Scheduler has been installed in the /opt/maestro directory, in a Bourne/Korn shell environment, the PATH variable should be defined as:

```
PATH=/opt/maestro:/opt/maestro/bin:$PATH. export PATH
```

In addition to the PATH, you must also set the TWS_TISDIR variable to TWShome. The TWS_TISDIR variable enables IBM Tivoli Workload Scheduler to display messages in the correct language and codeset, such as TWS_TISDIR=/opt/maestro. export TWS_TISDIR. In this way, the necessary environment variables and search paths are set to allow you to run commands, such as conman or composer commands, even if you are not located in the TWShome path. Alternatively, you can use the tws_env shell script to set up both the PATH and TWS_TISDIR variables. These variables must be set before you can run commands. The tws_env script has been provided in two versions:

- tws_env.sh for Bourne and Korn shell environments
- tws_env.csh for C Shell environments

To start the IBM Tivoli Workload Scheduler network management process (netman) automatically as a daemon each time you boot your system, add one of the following sets of code to the /etc/rc file, or the proper file for your system.

To start netman only:

Example 15-13 Start netman

```
if [-x twshome/StartUp]
then
echo “netman started...”
/bin/su - twsuser -c “ twshome/StartUp”
fi
```

To start the entire IBM Tivoli Workload Scheduler process tree:

Example 15-14 Tivoli Workload Scheduler process tree

```
if [-x twshome/bin/conman
then
echo “Workload Scheduler started...”
/bin/su - twsuser -c “ twshome/bin/conman start”
fi
```

15.2.3 Verify the Tivoli Workload Scheduler installation

To verify the installation, start the Tivoli Workload Scheduler and verify that it starts without any error messages.

If there are no active workstations in Tivoli Workload Scheduler for z/OS for the Tivoli Workload Scheduler agent, only the netman process will be started. But you can verify that the netman process is started and that it listens to the IP port number that you have decided to use in your end-to-end environment.

15.3 Define, activate, verify fault-tolerant workstations

To be able to define jobs in Tivoli Workload Scheduler for z/OS to be scheduled on FTWs, the workstations must be defined in Tivoli Workload Scheduler for z/OS controller.

The workstations that are defined via the CPUREC keyword should also be defined in the Tivoli Workload Scheduler for z/OS workstation database before they can be activated in the Tivoli Workload Scheduler for z/OS plan. The workstations are defined the same way as computer workstations in Tivoli Workload Scheduler for z/OS, except they need a special flag: *fault tolerant*. This flag is used to indicate in Tivoli Workload Scheduler for z/OS that these workstations should be treated as FTWs.

When the FTWs have been defined in the Tivoli Workload Scheduler for z/OS workstation database, they can be activated in the Tivoli Workload Scheduler for z/OS plan by either running a plan replan or plan extend batch job.

The process is as follows:

1. Create a CPUREC definition for the workstation as described in “CPUREC statement” on page 406.
2. Define the FTW in the Tivoli Workload Scheduler for z/OS workstation database. Remember to set it to **fault tolerant**.
3. Run Tivoli Workload Scheduler for z/OS plan replan or plan extend to activate the workstation definition in Tivoli Workload Scheduler for z/OS.
4. Verify that the FTW gets active and linked.
5. Define jobs and job streams on the newly created and activated FTW as described in 15.4, “Creating fault-tolerant workstation job definitions and job streams” on page 449.

Important: The order of the operations in this process is important.

15.3.1 Define fault-tolerant workstation in Tivoli Workload Scheduler controller workstation database

A fault-tolerant workstation can be defined either from Tivoli Workload Scheduler for z/OS ISPF dialogs (use option 1.1 from the main menu) or in the JSC.

The following steps show how to define an FTW from the JSC:

1. In the **Actions Lists**, under **New Workstation**, select the instance for the Tivoli Workload Scheduler for z/OS controller where the workstation should be defined (TWSC-zOS in our example).

   The Properties - Workstation in Database window opens (Figure 15-32).

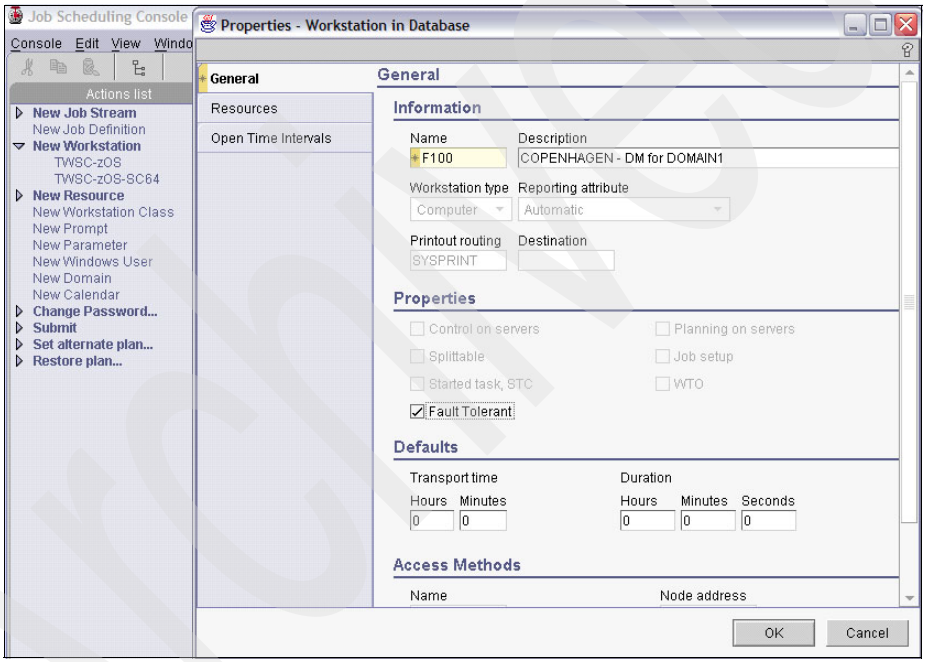


Figure 15-32 Defining a fault-tolerant workstation from the JSC

2. Select the **Fault Tolerant** check box and fill in the Name field (the four-character name of the FTW) and, optionally, the Description field.

> **Note:** Using the first part of the description field to list the DNS name or host name for the FTW makes it easier to remember which server or machine the four-character workstation name in Tivoli Workload Scheduler for z/OS relates to. The description field holds 32 alphanumeric characters.

3. Save the new workstation definition by clicking **OK**.

Note: When we used the JSC to create FTWs as described, we sometimes received this error:

```
GJSO027E Cannot save the workstation xxxx.
Reason: EQQW787E FOR FT WORKSTATIONS RESOURCES CANNOT BE USED
AT PLANNING
```

If you receive this error when creating the FTW from the JSC, then select the **Resources** tab (see Figure 15-32 on page 445) and un-check the **Used for planning** check box for Resource 1 and Resource 2. This must be done before selecting the Fault Tolerant check box on the General tab.

15.3.2 Activate the fault-tolerant workstation definition

Fault-tolerant workstation definitions can be activated in the Tivoli Workload Scheduler for z/OS plan either by running the replan or the extend plan programs in the Tivoli Workload Scheduler for z/OS controller.

When running the replan or extend program, Tivoli Workload Scheduler for z/OS creates (or re-creates) the Symphony file and distributes it to the domain managers at the first level. These domain managers, in turn, distribute the Symphony file to their subordinate fault-tolerant agents and domain managers, and so on. If the Symphony file is successfully created and distributed, all defined FTWs should be linked and active.

We run the replan program and verify that the Symphony file is created in the end-to-end server. We also verify that the FTWs become available and have linked status in the Tivoli Workload Scheduler for z/OS plan.

15.3.3 Verify that the fault-tolerant workstations are active and linked

Verify that no warning or error message is in the replan batch job (EQQMLOG). The message log should show that all topology statements (DOMREC, CPUREC, and USRREC) have been accepted without any errors or warnings.

Verify messages in plan batch job

For a successful creation of the Symphony file, the message log should show messages similar to those in Example 15-15.

Example 15-15 Plan batch job EQQMLOG messages when Symphony file is created

```
EQQZ014I MAXIMUM RETURN CODE FOR PARAMETER MEMBER TPDOMAIN IS: 0000
EQQZ013I NOW PROCESSING PARAMETER LIBRARY MEMBER TPUSER
```

```
EQQZ014I MAXIMUM RETURN CODE FOR PARAMETER MEMBER TPUSER   IS: 0000
EQQQ502I SPECIAL RESOURCE DATASPACE HAS BEEN CREATED.
EQQQ502I 00000020 PAGES ARE USED FOR 00000100 SPECIAL RESOURCE RECORDS.
EQQ3011I WORKSTATION F100 SET AS DOMAIN MANAGER FOR DOMAIN DM100
EQQ3011I WORKSTATION F200 SET AS DOMAIN MANAGER FOR DOMAIN DM200
EQQ3105I A NEW CURRENT PLAN (NCP) HAS BEEN CREATED
EQQ3106I WAITING FOR SCP
EQQ3107I SCP IS READY: START JOBS ADDITION TO SYMPHONY FILE
EQQ4015I RECOVERY JOB OF F100DJ01 HAS NO JOBWS KEYWORD SPECIFIED,
EQQ4015I THE WORKSTATION F100 OF JOB F100DJ01 IS USED
EQQ3108I JOBS ADDITION TO SYMPHONY FILE COMPLETED
EQQ3101I 0000019 JOBS ADDED TO THE SYMPHONY FILE FROM THE CURRENT PLAN
EQQ3087I SYMNEW FILE HAS BEEN CREATED
```

Verify messages in the end-to-end server message log

In the Tivoli Workload Scheduler for z/OS end-to-end server message log, we see the messages in Example 15-16. These messages show that the Symphony file has been created by the plan replan batch jobs and that it was possible for the end-to-end server to switch to the new Symphony file.

Example 15-16 End-to-end server messages when Symphony file is created

```
EQQPT30I Starting switching Symphony
EQQPT12I The Mailman process (pid=Unknown) ended successfully
EQQPT12I The Batchman process (pid=Unknown) ended successfully
EQQPT22I Input Translator thread stopped until new Symphony will be available
EQQPT31I Symphony successfully switched
EQQPT20I Input Translator waiting for Batchman and Mailman are started
EQQPT21I Input Translator finished waiting for Batchman and Mailman
EQQPT23I Input Translator thread is running
```

Verify messages in the controller message log

The Tivoli Workload Scheduler for z/OS controller shows the messages in Example 15-17 on page 447, which indicate that the Symphony file was created successfully and that the fault-tolerant workstations are active and linked.

Example 15-17 Controller messages when Symphony file is created

```
EQQN111I SYMNEW FILE HAS BEEN CREATED
EQQW090I THE NEW SYMPHONY FILE HAS BEEN SUCCESSFULLY SWITCHED
EQQWL10W WORK STATION F100, HAS BEEN SET TO LINKED STATUS
EQQWL10W WORK STATION F100, HAS BEEN SET TO ACTIVE   STATUS
EQQWL10W WORK STATION F101, HAS BEEN SET TO LINKED STATUS
EQQWL10W WORK STATION F102, HAS BEEN SET TO LINKED STATUS
```

```
EQQWL10W WORK STATION F101, HAS BEEN SET TO ACTIVE   STATUS
EQQWL10W WORK STATION F102, HAS BEEN SET TO ACTIVE   STATUS
```

Verify that fault-tolerant workstations are active and linked

After the replan job has completed and output messages have been displayed, the FTWs are checked using the JSC instance pointing to Tivoli Workload Scheduler for z/OS controller (Figure 15-33).

The Fault Tolerant column indicates that it is an FTW. The Linked column indicates whether the workstation is linked. The Status column indicates whether the mailman process is up and running on the FTW.

Status of all Workstations - TWSC-zOS

Name	Status	Internal Status	Fault Tolerant	Linked	Reporting Attribute	Type
F100	Available		Yes	Yes	Automatic	Computer
F101	Available		Yes	Yes	Automatic	Computer
F102	Available		Yes	Yes	Automatic	Computer
F200	Not Available	Offline	Yes	No	Automatic	Computer

Figure 15-33 Status of FTWs in the Tivoli Workload Scheduler for z/OS plan

The F200 workstation is `Not Available` because we have not installed a Tivoli Workload Scheduler fault-tolerant workstation on this machine yet. We have prepared for a future installation of the F200 workstation by creating the related CPUREC definitions for F200 and defined the FTW (F200) in the Tivoli Workload Scheduler controller workstation database.

Tip: If the workstation does not link as it should, the cause could be that the writer process has not initiated correctly or the run number for the Symphony file on the FTW is not the same as the run number on the master. Mark the unlinked workstations and right-click to open a pop-up menu where you can click **Link** to try to link the workstation.

The run number for the Symphony file in the end-to-end server can be seen from ISPF panels in option 6.6 from the main menu.

Figure 15-34 shows the status of the same FTWs, as it is shown in the JSC, when looking at the Symphony file at domain manager F100.

Much more information is available for each FTW. For example, in Figure 15-34 we can see that jobman and writer are running and that we can run 20 jobs in parallel on the FTWs (the Limit column). The information in the Run, CPU Type, and Domain columns is read from the Symphony file and generated by the plan

programs based on the specifications in CPUREC and DOMREC definitions. This is one of the reasons why we suggest activating support for JSC when running end-to-end scheduling with Tivoli Workload Scheduler for z/OS.

Note that the status of the OPCMASTER workstation is correct, and remember that the OPCMASTER workstation and the MASTERDM domain are predefined in Tivoli Workload Scheduler for z/OS and cannot be changed.

Jobman is not running on OPCMASTER (in USS in the end-to-end server), because the end-to-end server is not supposed to run jobs in USS. So the information that jobman is not running on the OPCMASTER workstation is valid.

Status of all Workstations - TWS

Name	Jobman Run	Link Status	Writer Run	Limit	Fence	Run	CPU Type	Domain
OPCMASTER	No	LINKED	Yes	0	0	237	MASTER	MASTERDM
F100	Yes			20	0	237	MANAGER	DM100
F101	Yes	LINKED	Yes	20	0	237	FTA	DM100
F102	Yes	LINKED	Yes	20	0	237	FTA	DM100
F200	No	UNLIN...	No	20	0	237	MANAGER	DM200

Figure 15-34 Status of FTWs in the Symphony file on domain manager F100

15.4 Creating fault-tolerant workstation job definitions and job streams

When the FTWs are active and linked in Tivoli Workload Scheduler for z/OS, you can run jobs on these workstations. To submit work to the FTWs in Tivoli Workload Scheduler for z/OS, you should:

1. Define the script (the JCL or the task) that should be executed on the FTW, (that is, on the server).

   When defining scripts in Tivoli Workload Scheduler for z/OS, the script can be placed central in the Tivoli Workload Scheduler for z/OS job library or non-centralized on the FTW (on the Tivoli Workload Scheduler server).

   Definitions of scripts are found in:

   – 15.4.1, “Centralized and non-centralized scripts” on page 450
   – 15.4.2, “Definition of centralized scripts” on page 452,
   – 15.4.3, “Definition of non-centralized scripts” on page 454
   – 15.4.4, “Combining centralized script and VARSUB and JOBREC” on page 465

2. Create a job stream (application) in Tivoli Workload Scheduler for z/OS and add the job (operation) defined in step 1 on page 449.

   It is possible to add the job (operation) to an existing job stream and create dependencies between jobs on FTWs and jobs on mainframe.

   Definition of FTW jobs and job streams in Tivoli Workload Scheduler for z/OS is found in 15.4.5, “Definition of FTW jobs and job streams in the controller” on page 466.

15.4.1 Centralized and non-centralized scripts

A job can use two kinds of scripts: centralized or non-centralized.

A centralized script is a script that resides in the controller job library (EQQJBLIB dd-card, also called JOBLIB) and that is downloaded to the FTW every time the job is submitted. Figure 15-35 illustrates the relationship between the centralized script job definition and member name in the job library (JOBLIB).

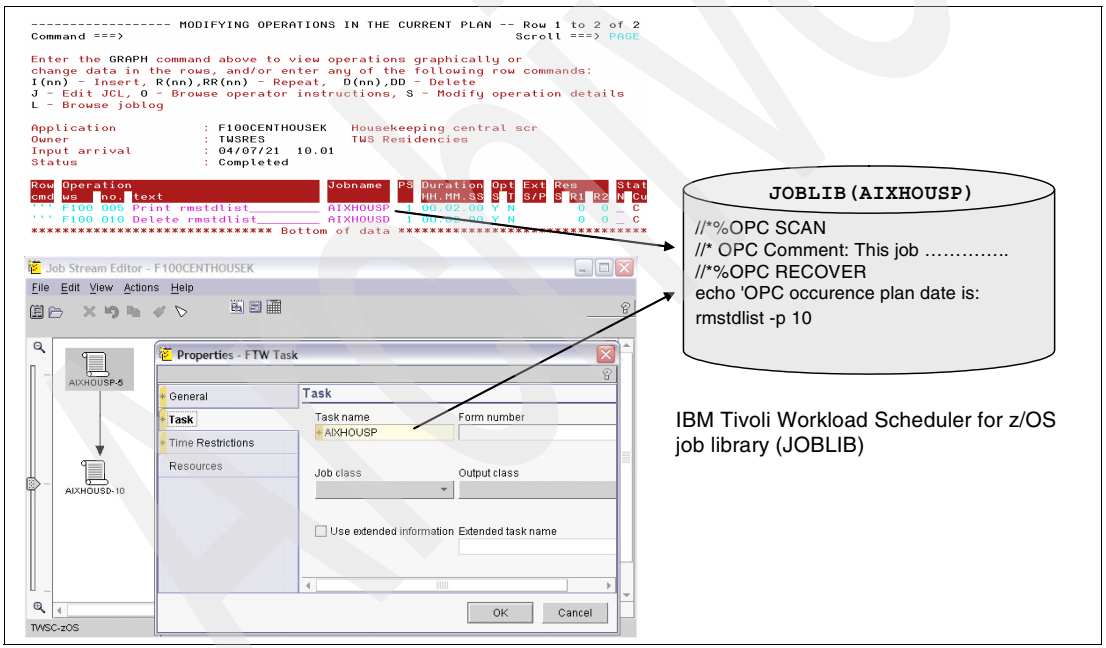


Figure 15-35 Centralized script defined in controller job library (JOBLIB)

A non-centralized script is a script that is defined in the SCRPTLIB and that resides on the FTW. Figure 15-36 shows the relationship between the job definition and the member name in the script library (EQQSCLIB).

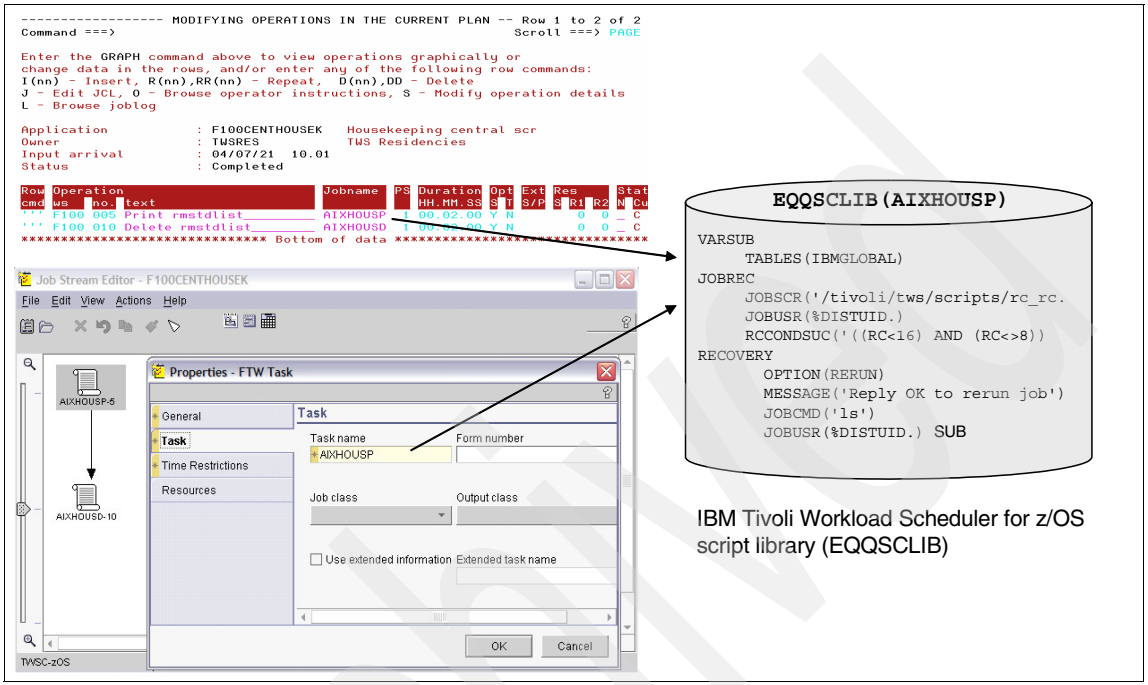


Figure 15-36 Non-centralized script defined in controller script library (EQQSCLIB)

15.4.2 Definition of centralized scripts

Define the centralized script job (operation) in a Tivoli Workload Scheduler for z/OS job stream (application) with the Centralized Script option set to Y (Yes). See Figure 15-37.

Note: The default is N (No) for all operations in Tivoli Workload Scheduler for z/OS.

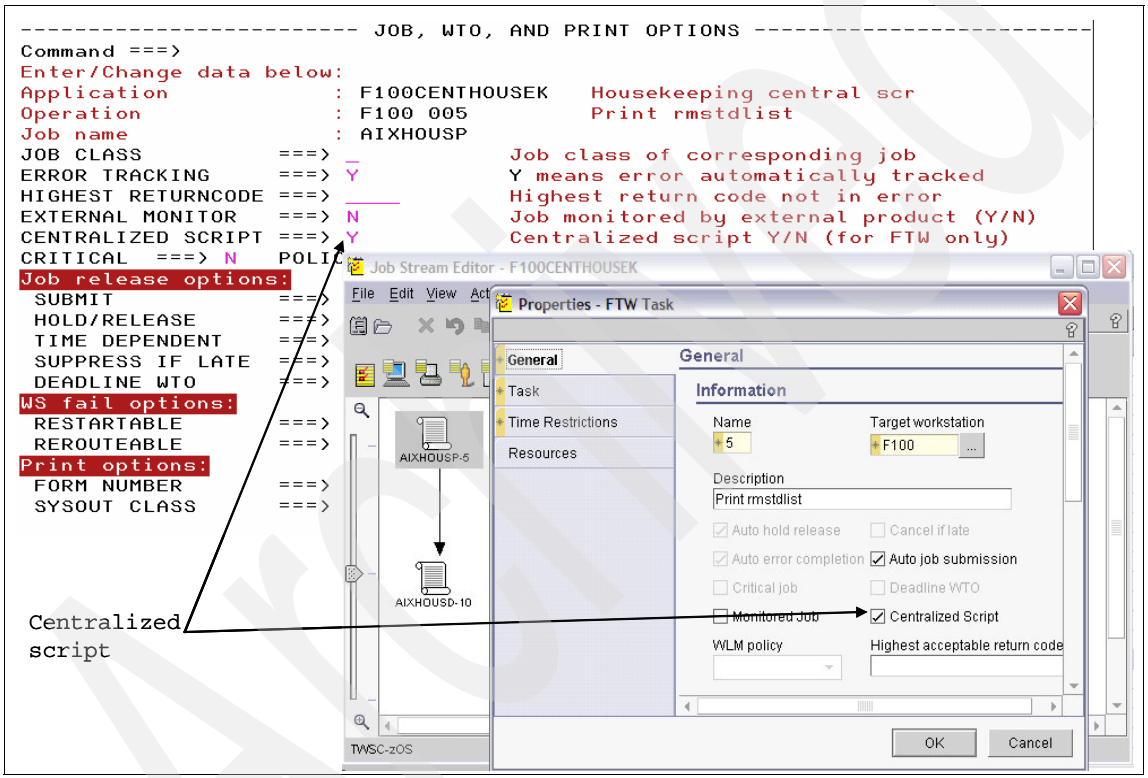


Figure 15-37 Centralized script option set in ISPF panel or JSC window

A centralized script is a script that resides in the Tivoli Workload Scheduler for z/OS JOBLIB and that is downloaded to the fault-tolerant agent every time the job is submitted.

The centralized script is defined the same way as a normal job JCL in Tivoli Workload Scheduler for z/OS.

The centralized script in Example 15-18 is running the rmstdlist program that is delivered with Tivoli Workload Scheduler. In the centralized script, we use Tivoli Workload Scheduler for z/OS Automatic Recovery as well as JCL variables.

Example 15-18 Centralized script for job AIXHOUSP defined in controller JOBLIB

```
EDIT       TWS.V8R20.JOBLIB(AIXHOUSP) - 01.02            Columns 00001 00072
 Command ===>                                              Scroll ===> CSR
 ****** ***************************** Top of Data ******************************
 000001 //*%OPC SCAN
 000002 //* OPC Comment: This job calls TWS rmstdlist script.
 000003 //* OPC ======== - The rmstdlist script is called with -p flag and
 000004 //* OPC            with parameter 10.
 000005 //* OPC          - This means that the rmstdlist script will print
 000006 //* OPC            files in the stdlist directory older than 10 days.
 000007 //* OPC          - If rmstdlist ends with RC in the interval from 1
 000008 //* OPC            to 128, OPC will add recovery application
 000009 //* OPC            F100CENTRECAPPL.
 000010 //* OPC
 000011 //*%OPC RECOVER JOBCODE=(1-128),ADDAPPL=(F100CENTRECAPPL),RESTART=(NO)
 000012 //* OPC
 000013 echo 'OPC occurrence plan date is: &ODMY1.'
 000014 rmstdlist -p 10
 ****** **************************** Bottom of Data ****************************
```

Rules when creating centralized scripts

Follow these rules when creating the centralized scripts in the Tivoli Workload Scheduler for z/OS JOBLIB:

- Each line starts in column 1 and ends in column 80.
- A backslash (\) in column 80 can be used to continue script lines with more than 80 characters.
- Blanks at the end of a line are automatically removed.
- Lines that start with //* OPC, //*%OPC, or //*>OPC are used for comments, variable substitution directives, and automatic job recovery. These lines are automatically removed before the script is downloaded to the FTA.

15.4.3 Definition of non-centralized scripts

Non-centralized scripts are defined in a special partitioned data set, EQQSCLIB, that is allocated in the Tivoli Workload Scheduler for z/OS controller started task procedure and used to store the job or task definitions for FTA jobs. The script (the JCL) resides on the fault-tolerant agent.

Note: This is the default behavior in Tivoli Workload Scheduler for z/OS for fault-tolerant agent jobs.

You must use the JOBREC statement in every SCRPTLIB member to specify the script or command to run. In the SCRPTLIB members, you can also specify the following statements:

- VARSUB to use the Tivoli Workload Scheduler for z/OS automatic substitution of variables when the Symphony file is created or when an operation on an FTW is added to the current plan dynamically.
- RECOVERY to use the Tivoli Workload Scheduler recovery.

Example 15-19 shows the syntax for the VARSUB, JOBREC, and RECOVERY statements.

Example 15-19 Syntax for VARSUB, JOBREC, and RECOVERY statements

```
VARSUB
   TABLES(GLOBAL|tab1,tab2,..|APPL)
   PREFIX('char')
   BACKPREF('char')
   VARFAIL(YES|NO)
   TRUNCATE(YES|NO)
JOBREC
   JOBSCR|JOBCMD ('task')
   JOBUSR ('username')
   INTRACTV(YES|NO)
   RCCONDSUC('success condition')
RECOVERY
   OPTION(STOP|CONTINUE|RERUN)
   MESSAGE('message')
   JOBCMD|JOBSCR('task')
   JOBUSR ('username')
   JOBWS('wsname')
   INTRACTV(YES|NO)
   RCCONDSUC('success condition')
```

If you define a job with a SCRPTLIB member in the Tivoli Workload Scheduler for z/OS database that contains errors, the daily planning batch job sets the status of that job to `failed` in the Symphony file. This change of status is not shown in the Tivoli Workload Scheduler for z/OS interface. You can find the messages that explain the error in the log of the daily planning batch job.

If you dynamically add a job to the plan in Tivoli Workload Scheduler for z/OS whose associated SCRPTLIB member contains errors, the job is not added. You can find the messages that explain this failure in the controller EQQMLOG.

Rules when creating JOBREC, VARSUB, or RECOVERY statements

Each statement consists of a statement name, keywords, and keyword values, and follows TSO command syntax rules. When you specify SCRPTLIB statements, follow these rules:

- Statement data must be in columns 1 through 72. Information in columns 73 through 80 is ignored.
- A blank serves as the delimiter between two keywords; if you supply more than one delimiter, the extra delimiters are ignored.
- Continuation characters and blanks are not used to define a statement that continues on the next line.
- Values for keywords are enclosed in parentheses. If a keyword can have multiple values, the list of values must be separated by valid delimiters. Delimiters are not allowed between a keyword and the left parenthesis of the specified value.
- Type /* to start a comment and */ to end a comment. A comment can span record images in the parameter member and can appear anywhere *except* in the middle of a keyword or a specified value.
- A statement continues until the next statement or until the end of records in the member.
- If the value of a keyword includes spaces, enclose the value within single or double quotation marks as in Example 15-20.

Example 15-20 JOBCMD and JOBSCR examples

```
JOBCMD('ls la')
JOBSCR('C:/USERLIB/PROG/XME.EXE')
JOBSCR("C:/USERLIB/PROG/XME.EXE")
JOBSCR("C:/USERLIB/PROG/XME.EXE 'THIS IS THE PARAMETER LIST' ")
JOBSCR('C:/USERLIB/PROG/XME.EXE "THIS IS THE PARAMETER LIST" ')
```

Description of the VARSUB statement

The VARSUB statement defines the variable substitution options. This statement must always be the first one in the members of the SCRPTLIB. For more information about the variable definition, see *IBM Tivoli Workload Scheduler for z/OS Managing the Workload*, Version 8.2 (Maintenance Release April 2004), SC32-1263.

Note: Can be used in combination with a job that is defined with a centralized script.

Figure 15-38 shows the format of the VARSUB statement.

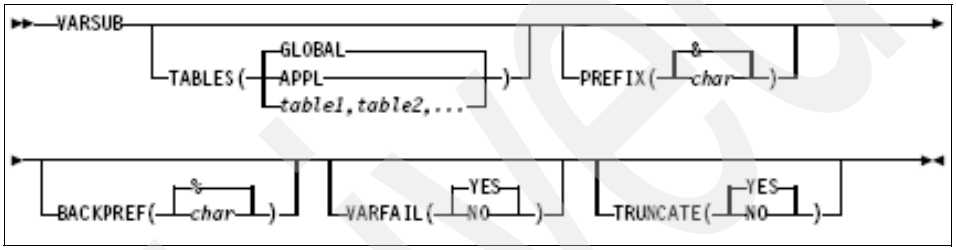


Figure 15-38 Format of the VARSUB statement

VARSUB is defined in the members of the EQQSCLIB library, as specified by the EQQSCLIB DD of the Tivoli Workload Scheduler for z/OS controller and the plan extend, replan, and Symphony renew batch job JCL.

Description of the VARSUB parameters

VARSUB parameters can be described as follows:

- TABLES(GLOBAL|APPL|*table1,table2,...*)

  Identifies the variable tables that must be searched and the search order. APPL indicates the application variable table (see the VARIABLE TABLE field in the MCP panel, at Occurrence level). GLOBAL indicates the table defined in the GTABLE keyword of the OPCOPTS controller and BATCHOPT batch options.

- PREFIX(*char*|&)

  A non-alphanumeric character that precedes a variable. It serves the same purpose as the ampersand (&) character that is used in variable substitution in z/OS JCL.

- BACKPREF(*char*|%)

  A non-alphanumeric character that delimits a variable to form simple and compound variables. It serves the same purpose as the percent (%) character that is used in variable substitution in z/OS JCL.

- VARFAIL(NO|YES)

  Specifies whether Tivoli Workload Scheduler for z/OS is to issue an error message when a variable substitution error occurs. If you specify NO, the variable string is left unchanged without any translation.

- TRUNCATE(YES|NO)

  Specifies whether variables are to be truncated if they are longer than the allowed length. If you specify NO and the keywords are longer than the allowed length, an error message is issued. The allowed length is the length of the keyword for which you use the variable. For example, if you specify a variable of five characters for the JOBWS keyword, the variable is truncated to the first four characters.

Description of the JOBREC statement

The JOBREC statement defines the fault-tolerant workstation job properties. You must specify JOBREC for each member of the SCRPTLIB. For each job this statement specifies the script or the command to run and the user who must run the script or command.

Note: JOBREC can be used in combination with a job that is defined with a centralized script.

Figure 15-39 shows the format of the JOBREC statement.

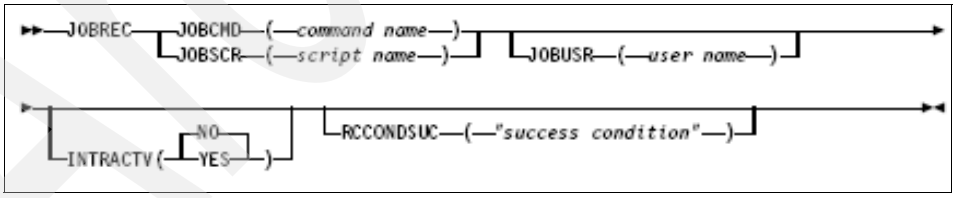


Figure 15-39 Format of the JOBREC statement

JOBREC is defined in the members of the EQQSCLIB library, as specified by the EQQSCLIB DD of the Tivoli Workload Scheduler for z/OS controller and the plan extend, replan, and Symphony renew batch job JCL.

Description of the JOBREC parameters

JOBREC parameters can be described as follows:

- JOBSCR*(script name)*

  Specifies the name of the shell script or executable file to run for the job. The maximum length is 4095 characters. If the script includes more than one word, it must be enclosed within single or double quotation marks. Do not specify this keyword if the job uses a centralized script.

- JOBCMD*(command name)*

  Specifies the name of the shell command to run the job. The maximum length is 4095 characters. If the command includes more than one word, it must be enclosed in single or double quotation marks. Do not specify this keyword if the job uses a centralized script.

- JOBUSR*(user name)*

  Specifies the name of the user submitting the specified script or command. The maximum length is 47 characters. If you do not specify the user in the JOBUSR keyword, the user defined in the CPUUSER keyword of the CPUREC statement is used. The CPUREC statement is the one related to the workstation on which the specified script or command must run. If the user is not specified in the CPUUSER keyword, the tws user is used.

  If the script is centralized, you can also use the job-submit exit (EQQUX001) to specify the user name. This user name overrides the value specified in the JOBUSR keyword. In turn, the value that is specified in the JOBUSR keyword overrides that specified in the CPUUSER keyword of the CPUREC statement. If no user name is specified, the tws user is used.

  If you use this keyword to specify the name of the user who submits the specified script or command on a Windows fault-tolerant workstation, you must associate this user name to the Windows workstation in the USRREC initialization statement.

- INTRACTV(YES|NO)

  Specifies that a Windows job runs interactively on the Windows desktop. This keyword is used only for jobs running on Windows fault-tolerant workstations.

- RCCONDSUC("*success condition*")

  An expression that determines the return code (RC) that is required to consider a job as successful. If you do not specify this keyword, the return code equal to zero corresponds to a successful condition. A return code different from zero corresponds to the job abend.

  The success condition maximum length is 256 characters and the total length of JOBCMD or JOBSCR plus the success condition must be 4086 characters. This is because the TWSRCMAP string is inserted between the success

condition and the script or command name. For example, the **dir** command together with the success condition RC<4 is translated into:

```
dir TWSRCMAP: RC<4
```

The success condition expression can contain a combination of comparison and Boolean expressions:

– *Comparison expression* specifies the job return codes. The syntax is:

(*RC operator operand*)

- *RC* is the RC keyword (type RC).
- *operator* is the comparison operator. It can have the values shown in Table 15-6.

Table 15-6 Comparison operators

Example	Operator	Description
RC < a	<	Less than
RC <= a	<=	Less than or equal to
RC > a	>	Greater than
RC >= a	>=	Greater than or equal to
RC = a	=	Equal to
RC <> a	<>	Not equal to

- *operand* is an integer between -2147483647 and 2147483647.

For example, you can define a successful job as a job that ends with a return code less than or equal to 3 as follows:

```
RCCONDSUC "(RC <= 3)"
```

– *Boolean expression* specifies a logical combination of comparison expressions. The syntax is:

comparison_expression operator comparison_expression

- *comparison_expression*
  The expression is evaluated from left to right. You can use parentheses to assign a priority to the expression evaluation.
- *operator*
  Logical operator. It can have the following values: and, or, not.

For example, you can define a successful job as a job that ends with a return code less than or equal to 3 or with a return code not equal to 5, and less than 10 as follows:

```
RCCONDSUC "(RC<=3) OR ((RC<>5) AND (RC<10))"
```

Description of the RECOVERY statement

Scheduler recovery for a job whose status is in error, but whose error code is not FAIL. To run the recovery, you can specify one or both of the following recovery actions:

- A recovery job (JOBCMD or JOBSCR keywords)
- A recovery prompt (MESSAGE keyword)

The recovery actions must be followed by one of the recovery options (the OPTION keyword), STOP, CONTINUE, or RERUN. The default is stop with no recovery job and no recovery prompt. For more information about recovery in a distributed network, see *Tivoli Workload Scheduler Reference Guide Version 8.2* (Maintenance Release April 2004),SC32-1274.

The RECOVERY statement is ignored if it is used with a job that runs a centralized script.

Figure 15-40 shows the format of the RECOVERY statement.

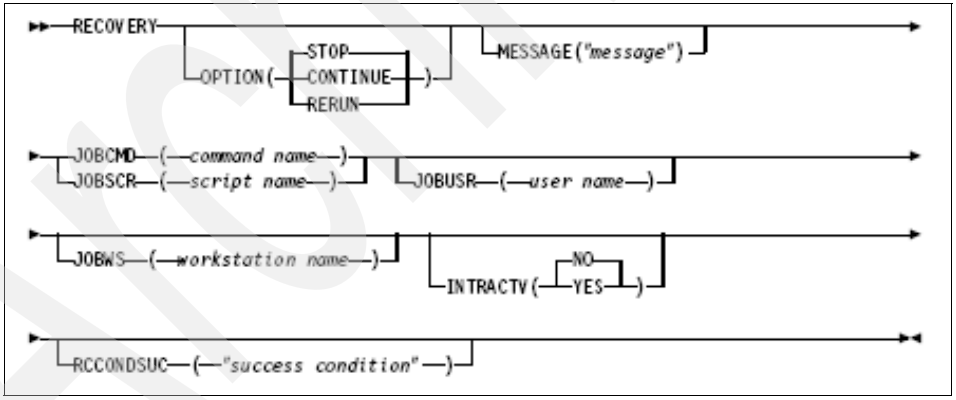


Figure 15-40 Format of the RECOVERY statement

RECOVERY is defined in the members of the EQQSCLIB library, as specified by the EQQSCLIB DD of the Tivoli Workload Scheduler for z/OS controller and the plan extend, replan, and Symphony renew batch job JCL.

Description of the RECOVERY parameters

The RECOVERY parameters can be described as follows:

- OPTION(STOP|CONTINUE|RERUN)

  Specifies the option that Tivoli Workload Scheduler for z/OS must use when a job abends. For every job, Tivoli Workload Scheduler for z/OS enables you to define a recovery option. You can specify one of the following values:

  - STOP**:** Do not continue with the next job. The current job remains in error. You cannot specify this option if you use the MESSAGE recovery action.
  - CONTINUE: Continue with the next job. The current job status changes to complete in the z/OS interface.
  - RERUN**:** Automatically rerun the job (once only). The job status changes to ready, and then to the status of the rerun. Before rerunning the job for a second time, an automatically generated recovery prompt is displayed.

- MESSAGE**(**"*message*"**)**

  Specifies the text of a recovery prompt, enclosed in single or double quotation marks, to be displayed if the job abends. The text can contain up to 64 characters. If the text begins with a colon (:), the prompt is displayed, but no reply is required to continue processing. If the text begins with an exclamation mark (!), the prompt is not displayed but a reply is required to proceed. You cannot use the recovery prompt if you specify the recovery STOP option without using a recovery job.

- JOBCMD*(command name)*

  Specifies the name of the shell command to run if the job abends. The maximum length is 4095 characters. If the command includes more than one word, it must be enclosed in single or double quotation marks.

- JOBSCR*(script name)*

  Specifies the name of the shell script or executable file to be run if the job abends. The maximum length is 4095 characters. If the script includes more than one word, it must be enclosed in single or double quotation marks.

- JOBUSR*(user name)*

  Specifies the name of the user submitting the recovery job action. The maximum length is 47 characters. If you do not specify this keyword, the user defined in the JOBUSR keyword of the JOBREC statement is used. Otherwise, the user defined in the CPUUSER keyword of the CPUREC statement is used. The CPUREC statement is the one related to the workstation on which the recovery job must run. If the user is not specified in the CPUUSER keyword, the tws user is used.

If you use this keyword to specify the name of the user who runs the recovery on a Windows fault-tolerant workstation, you must associate this user name to the Windows workstation in the USRREC initialization statement

- JOBWS*(workstation name)*

  Specifies the name of the workstation on which the recovery job or command is submitted. The maximum length is four characters. The workstation must belong to the same domain as the workstation on which the main job runs. If you do not specify this keyword, the workstation name of the main job is used.

- INTRACTV(YES|NO)

  Specifies that the recovery job runs interactively on a Windows desktop. This keyword is used only for jobs running on Windows fault-tolerant workstations.

- RCCONDSUC**(**"*success condition*"**)**

  An expression that determines the return code (RC) that is required to consider a recovery job as successful. If you do not specify this keyword, the return code equal to zero corresponds to a successful condition. A return code different from zero corresponds to the job abend.

  The *success condition* maximum length is 256 characters and the total length of the JOBCMD or JOBSCR plus the success condition must be 4086 characters. This is because the TWSRCMAP string is inserted between the success condition and the script or command name. For example, the **dir** command together with the success condition RC<4 is translated into:

  ```
  dir TWSRCMAP: RC<4
  ```

  The *success condition* expression can contain a combination of comparison and Boolean expressions:

  – *Comparison expression* Specifies the job return codes. The syntax is:

    (*RC operator operand*)

    - *RC* is the RC keyword (type RC).
    - *operator* is the comparison operator. It can have the values in Table 15-7.

    Table 15-7 Operator comparison operator values

Example	Operator	Description
RC < a	<	Less than
RC <= a	<=	Less than or equal to
RC > a	>	Greater than
RC >= a	>=	Greater than or equal to

Example	Operator	Description
RC = a	=	Equal to
RC <> a	<>	Not equal to

- *operand* is an integer between -2147483647 and 2147483647.

For example, you can define a successful job as a job that ends with a return code less than or equal to 3 as:

```
RCCONDSUC "(RC <= 3)"
```

– Boolean expression: Specifies a logical combination of comparison expressions. The syntax is:

comparison_expression operator comparison_expression

- *comparison_expression* The expression is evaluated from left to right. You can use parentheses to assign a priority to the expression evaluation.
- *operator* Logical operator (it could be either: and, or, not).

For example, you can define a successful job as a job that ends with a return code less than or equal to 3 or with a return code not equal to 5, and less than 10 as follows:

```
RCCONDSUC "(RC<=3) OR ((RC<>5) AND (RC<10))"
```

Example VARSUB, JOBREC, and RECOVERY

For the test of VARSUB, JOBREC, and RECOVERY, we used the non-centralized script member as shown in Example 15-21.

Example 15-21 Non-centralized AIX script with VARSUB, JOBREC, and RECOVERY

```
EDIT       TWS.V8R20.SCRPTLIB(F100DJ02) - 01.05            Columns 00001 00072
Command ===>                                                  Scroll ===> CSR
****** **************************** Top of Data ******************************
000001 /* Definition for job with "non-centralized" script                   */
000002 /* ------------------------------------------------                   */
000003 /* VARSUB - to manage JCL variable substitution                       */
000004 VARSUB
000005      TABLES(E2EVAR)
000006      PREFIX('&')
000007      BACKPREF('%')
000008      VARFAIL(YES)
000009      TRUNCATE(YES)
000010 /* JOBREC - to define script, user and some other specifications      */
000011 JOBREC
```

```
000012      JOBCMD('rm &TWSHOME/demo.sh')
000013      JOBUSR ('%TWSUSER')
000014 /* RECOVERY - to define what FTA should do in case of error in job    */
000015 RECOVERY
000016      OPTION(RERUN)                     /* Rerun the job after recover*/
000017      JOBCMD('touch &TWSHOME/demo.sh')  /* Recover job                */
000018      JOBUSR('&TWSUSER')                /* User for recover job       */
000019      MESSAGE ('Create demo.sh on FTA?') /* Prompt message            */
****** **************************** Bottom of Data ****************************
```

The member F100DJ02 in the previous example was created in the SCRPTLIB (EQQSCLIB) partitioned data set. In the non-centralized script F100DJ02, we use VARSUB to specify how we want Tivoli Workload Scheduler for z/OS to scan for JCL variables and substitute JCL variables. The JOBREC parameters specify that we will run the UNIX (AIX) **rm** command for a file named demo.sh.

If the file does not exist (it will not exist the first time the script is run), run the recovery command (**touch**) that will create the missing file so that the JOBREC JOBCMD() can be rerun (OPTION(RERUN)) without any errors.

Before the job is rerun, reply yes to the message: Create demo.sh on FTA?

Example 15-22 shows another example. The job will be marked complete if return code from the script is less than 16 and different from 8 or equal to 20.

Example 15-22 Non-centralized script definition with RCCONDSUC parameter

```
EDIT       TWS.V8R20.SCRPTLIB(F100DJ03) - 01.01            Columns 00001 00072
Command ===>                                                 Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 /* Definition for job with "distributed" script                        */
000002 /* ------------------------------------------                         */
000003 /* VARSUB - to manage JCL variable substitution                        */
000004 VARSUB
000005        TABLES(IBMGLOBAL)
000006        PREFIX(%)
000007        VARFAIL(YES)
000008        TRUNCATE(NO)
000009 /* JOBREC - to define script, user and some other specifications      */
000010 JOBREC
000011        JOBSCR('/tivoli/tws/scripts/rc_rc.sh 12')
000012        JOBUSR(%DISTUID.)
000013        RCCONDSUC('((RC<16) AND (RC<>8)) OR (RC=20)')
```

Important: Be careful with lowercase and uppercase. In Example 15-22 on page 464, it is important that the variable name DISTUID is typed with capital letters because Tivoli Workload Scheduler for z/OS JCL variable names are always uppercase. On the other hand, it is important that the value for the DISTUID variable is defined in Tivoli Workload Scheduler for z/OS variable table IBMGLOBAL with lowercase letters, because the user ID is defined on the UNIX system with lowercase letters.

Be sure to type with *caps off* when editing members in SCRPTLIB (EQQSCLIB) for jobs with non-centralized scripts and members in Tivoli Workload Scheduler for z/OS JOBLIB (EQQJBLIB) for jobs with centralized scripts.

15.4.4 Combining centralized script and VARSUB and JOBREC

Sometimes it can be necessary to create a member in the EQQSCLIB (normally used for non-centralized script definitions) for a job that is defined in Tivoli Workload Scheduler for z/OS with a centralized script.

This can be the case if:

- The RCCONDSUC parameter will be used for the job to accept specific return codes or return code ranges.

  Note: You cannot use Tivoli Workload Scheduler for z/OS highest return code for fault-tolerant workstation jobs. You have to use the RCCONDSUC parameter.

- A special user should be assigned to the job with the JOBUSR parameter.
- Tivoli Workload Scheduler for z/OS JCL variables should be used in the JOBUSR() or the RCCONDSUC() parameters (for example).

Remember that the RECOVERY statement cannot be specified in EQQSCLIB for jobs with a centralized script. (It will be ignored.)

To make this combination, you simply:

1. Create the centralized script in Tivoli Workload Scheduler for z/OS JOBLIB. The member name should be the same as the job name defined for the operation (job) in the Tivoli Workload Scheduler for z/OS job stream (application).
2. Create the corresponding member in the EQQSCLIB. The member name should be the same as the member name for the job in the JOBLIB.

For example:

We have a job with a centralized script. In the job we should accept return codes less than 7 and the job should run with user `dbprod`.

To accomplish this, we define the centralized script in Tivoli Workload Scheduler for z/OS as shown in Example 15-18 on page 453. Next, we create a member in the EQQSCLIB with the same name as the member name used for the centralized script.

This member should contain only the JOBREC RCCONDSUC() and JOBUSR() parameters (Example 15-23).

Example 15-23 EQQSCLIB (SCRIPTLIB) definition for job with centralized script

```
EDIT       TWS.V8R20.SCRPTLIB(F100CJ02) - 01.05            Columns 00001 00072
Command ===>                                                 Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 JOBREC
000002         RCCONDSUC('RC<7')
000003         JOBUSR(dbprod)
****** **************************** Bottom of Data ****************************
```

15.4.5 Definition of FTW jobs and job streams in the controller

When the script is defined either as centralized in the Tivoli Workload Scheduler for z/OS job library (JOBLIB) or as non-centralized in the Tivoli Workload Scheduler for z/OS script library (EQQSCLIB), you can define some job streams (applications) to run the defined scripts.

Definition of job streams (applications) for fault-tolerant workstation jobs is done exactly the same way as normal mainframe job streams: The job is defined in the job stream, and dependencies are added (predecessor jobs, time dependencies, special resources). Optionally, a run cycle can be added to run the job stream at a set time.

When the job stream is defined, the fault-tolerant workstation jobs can be executed and the final verification test can be performed.

Figure 15-41 on page 467 shows an example of a job stream that is used to test the end-to-end scheduling environment. There are four distributed jobs (seen in the left window in the figure), and these jobs will run on workdays (seen in the right window).

It is not necessary to create a run cycle for job streams to test the FTW jobs, as they can be added manually to the plan in Tivoli Workload Scheduler for z/OS.

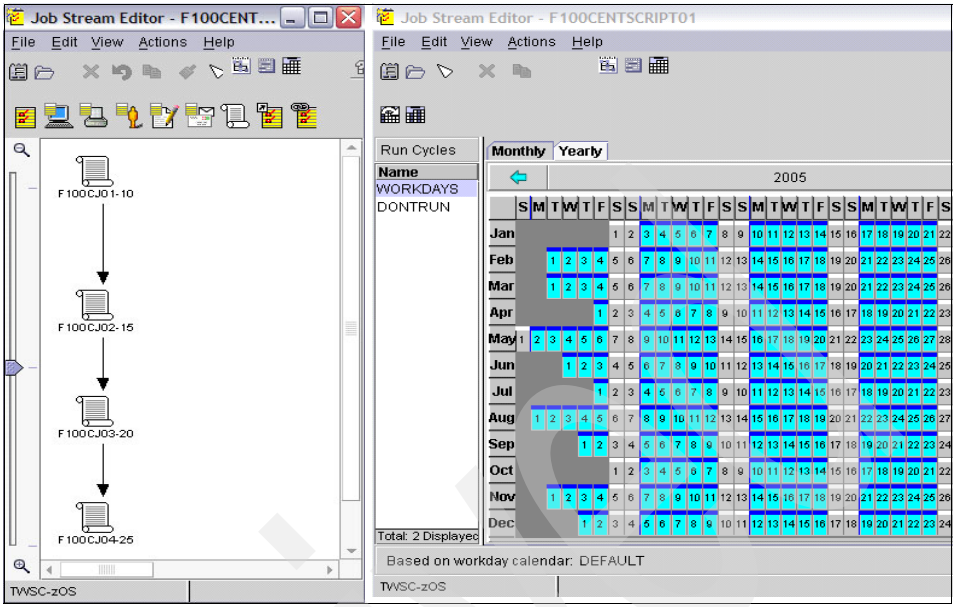


Figure 15-41 Example of a job stream used to test end-to-end scheduling

15.5 Verification test of end-to-end scheduling

At this point we have:

- Installed and configured the Tivoli Workload Scheduler for z/OS controller for end-to-end scheduling
- Installed and configured the Tivoli Workload Scheduler for z/OS end-to-end server
- Defined the network topology for the distributed Tivoli Workload Scheduler network in the end-to-end server and plan batch jobs
- Installed and configured Tivoli Workload Scheduler on the servers in the network for end-to-end scheduling
- Defined fault-tolerant workstations and activated these workstations in the Tivoli Workload Scheduler for z/OS network
- Verified that the plan program executed successfully with the end-to-end topology statements
- Created members with centralized and non-centralized scripts
- Created job streams containing jobs with centralized and non-centralized scripts

Now perform the final verification test of end-to-end scheduling to verify that:

- Jobs with centralized script definitions can be executed on the FTWs, and the job log can be browsed for these jobs.
- Jobs with non-centralized script definitions can be executed on the FTWs, and the job log can be browsed for these jobs.
- Jobs with a combination of centralized and non-centralized script definitions can be executed on the FTWs, and the job log can be browsed for these jobs.

The verification can be performed in several ways. Because we would like to verify that our end-to-end environment is working and that it is possible to run jobs on the FTWs, we have focused on this verification.

We used the Job Scheduling Console in combination with Tivoli Workload Scheduler for z/OS ISPF panels for the verifications. Of course, it is possible to perform the complete verification only with the ISPF panels.

Finally, if you decide to use only centralized scripts or non-centralized scripts, you do not have to verify both cases.

15.5.1 Verification of job with centralized script definitions

Now we add a job stream with a job defined with a centralized script using the job from Example 15-18 on page 453. Before the job was submitted, the JCL (script) was edited and the parameter on the rmstdlist program was changed from 10 to 1 (Figure 15-42).

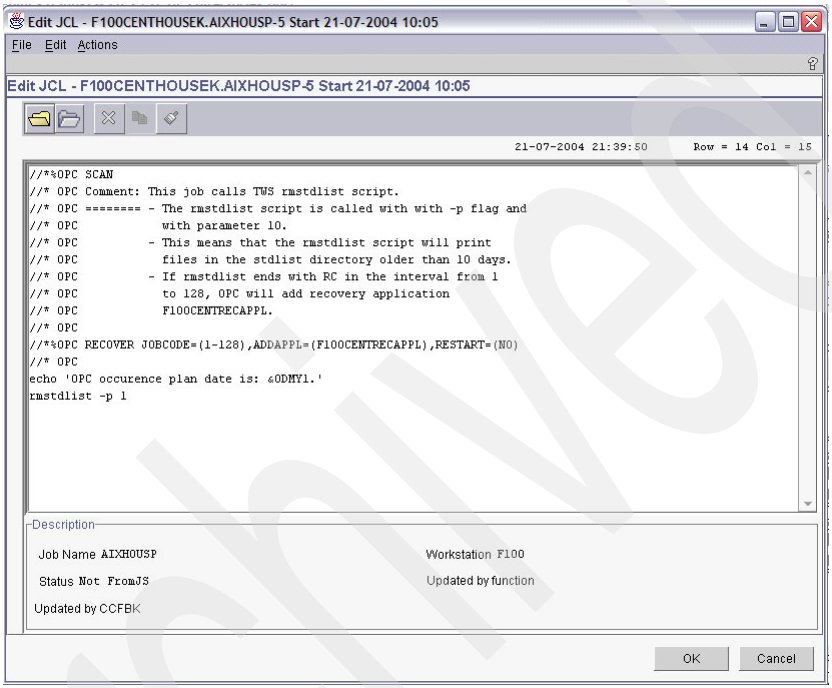


Figure 15-42 Edit JCL for centralized script, rmstdlist parameter changed from 10 to 1

The job is submitted, and it is verified that the job completes successfully on the FTA. Output is verified by doing browse job log. Figure 15-43 on page 470 shows only the first part of the job log. See the complete job log in Example 15-24 on page 470.

From the job log, you can see that the centralized script that was defined in the controller JOBLIB is copied to (see the line with the `= JCLFILE` text):

```
/tivoli/tws/twstest/tws/centralized/OPCMASTER.BB8CFD2B8A25EC41.J_0
05_F100CENTHOUSEK.sh
```

The Tivoli Workload Scheduler for z/OS JCL variable &ODMY1 in the “echo” line (Figure 15-42) has been substituted by the Tivoli Workload Scheduler for z/OS controller with the job stream planning date (for our case, 210704, seen in Example 15-24 on page 470).

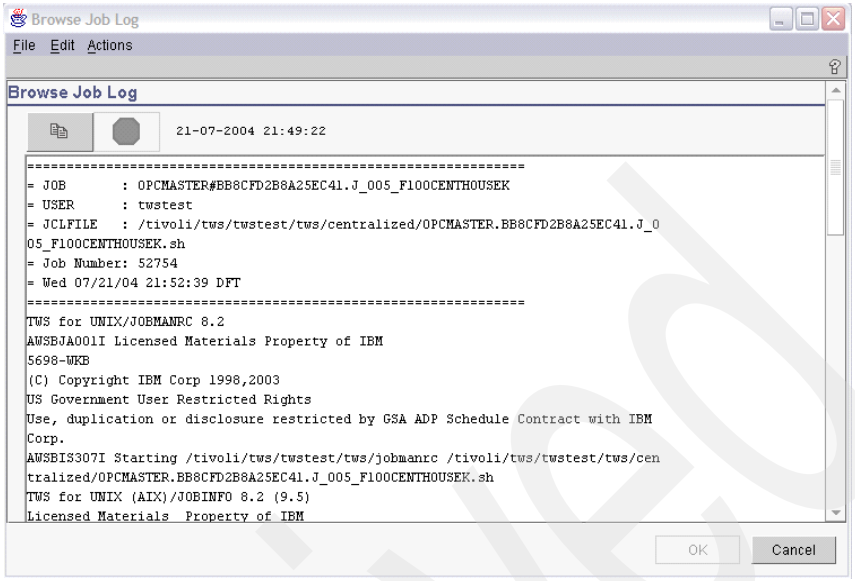


Figure 15-43 Browse first part of job log for the centralized script job in JSC

Example 15-24 The complete job log for the centralized script job

```
================================================================
= JOB       : OPCMASTER#BB8CFD2B8A25EC41.J_005_F100CENTHOUSEK
= USER      : twstest
= JCLFILE   : /tivoli/tws/twstest/tws/centralized/OPCMASTER.BB8CFD2B8A25EC41.J_0
05_F100CENTHOUSEK.sh
= Job Number: 52754
= Wed 07/21/04 21:52:39 DFT
================================================================
TWS for UNIX/JOBMANRC 8.2
AWSBJA001I Licensed Materials Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998,2003
US Government User Restricted Rights
Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
AWSBIS307I Starting /tivoli/tws/twstest/tws/jobmanrc /tivoli/tws/twstest/tws/cen
tralized/OPCMASTER.BB8CFD2B8A25EC41.J_005_F100CENTHOUSEK.sh
TWS for UNIX (AIX)/JOBINFO 8.2 (9.5)
Licensed Materials  Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998,2001
US Government User Restricted Rights
```

```
Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
Installed for user ''.
Locale LANG set to "C"
Now we are running the script /tivoli/tws/twstest/tws/centralized/OPCMASTER.BB8C
FD2B8A25EC41.J_005_F100CENTHOUSEK.sh
OPC occurrence plan date is: 210704
TWS for UNIX/RMSTDLIST 8.2
AWSBJA001I Licensed Materials Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998,2003
US Government User Restricted Rights
Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
AWSBIS324I Will list directories older than -1
/tivoli/tws/twstest/tws/stdlist/2004.07.13
/tivoli/tws/twstest/tws/stdlist/2004.07.14
/tivoli/tws/twstest/tws/stdlist/2004.07.15
/tivoli/tws/twstest/tws/stdlist/2004.07.16
/tivoli/tws/twstest/tws/stdlist/2004.07.18
/tivoli/tws/twstest/tws/stdlist/2004.07.19
/tivoli/tws/twstest/tws/stdlist/logs/20040713_NETMAN.log
/tivoli/tws/twstest/tws/stdlist/logs/20040713_TWSMERGE.log
/tivoli/tws/twstest/tws/stdlist/logs/20040714_NETMAN.log
/tivoli/tws/twstest/tws/stdlist/logs/20040714_TWSMERGE.log
/tivoli/tws/twstest/tws/stdlist/logs/20040715_NETMAN.log
/tivoli/tws/twstest/tws/stdlist/logs/20040715_TWSMERGE.log
/tivoli/tws/twstest/tws/stdlist/logs/20040716_NETMAN.log
/tivoli/tws/twstest/tws/stdlist/logs/20040716_TWSMERGE.log
/tivoli/tws/twstest/tws/stdlist/logs/20040718_NETMAN.log
/tivoli/tws/twstest/tws/stdlist/logs/20040718_TWSMERGE.log
===============================================================
= Exit Status          : 0
= System Time (Seconds) : 1     Elapsed Time (Minutes) : 0
= User Time (Seconds)   : 0
= Wed 07/21/04 21:52:40 DFT
===============================================================
```

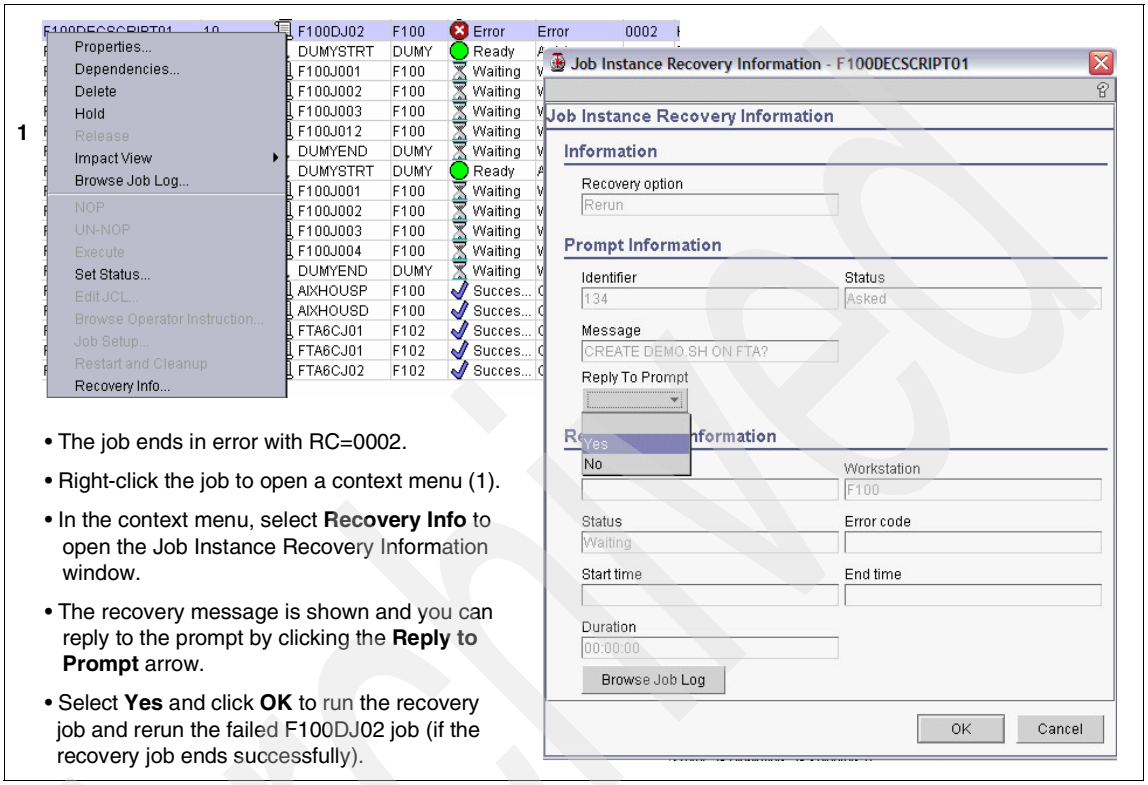
This completes the verification of the centralized script.

15.5.2 Verification of job with non-centralized scripts

Add a job stream with a job defined with a non-centralized script. Our example uses the non-centralized job script from Example 15-22 on page 464.

The job is submitted, and it is verified that the job ends in error. (Remember that the JOBCMD will try to remove a non-existing file.)

Reply to the prompt with **Yes** (Figure 15-44), and the recovery job is executed.



- The job ends in error with RC=0002.
- Right-click the job to open a context menu (1).
- In the context menu, select **Recovery Info** to open the Job Instance Recovery Information window.
- The recovery message is shown and you can reply to the prompt by clicking the **Reply to Prompt** arrow.
- Select **Yes** and click **OK** to run the recovery job and rerun the failed F100DJ02 job (if the recovery job ends successfully).

Figure 15-44 Running F100DJ02 job with non-centralized script and RECOVERY options

The same process can be performed in Tivoli Workload Scheduler for z/OS ISPF panels.

When the job ends in error, type `RI` (for Recovery Info) for the job in the Tivoli Workload Scheduler for z/OS Error list to get the panel shown in Figure 15-45 on page 473.

```
EQQREINP ------------------ OPERATION RECOVERY INFO ---------------------------
Option ===> _

Enter any of the following row commands:
L   - Browse joblog    PY   - Prompt reply Yes    PN   - Prompt reply No

Recovered job info
Application              : F100DECSCRIPT01    04/07/21  09.00
Operation                : F100 10            UNX23376

Recovery Option          : Rerun
Recovery Action          : Prompt / Job

Recovery job info
Recovery jobid           :                Duration:
Work Station Name        : F100           Status  : Waiting
Recovery job start/end   :                    /

Recovery prompt info
Prompt Id : 3134                          Status  : Asked
Message   : CREATE DEMO.SH ON FTA?
```

Figure 15-45 Recovery Info ISPF panel in Tivoli Workload Scheduler for z/OS

To reply Yes to the prompt, type `PY` in the Option field.

Then press Enter several times to see the result of the recovery job in the same panel. The Recovery job info fields will be updated with information for Recovery jobid, Duration, and so on (Figure 15-46).

```
EQQREINP ------------------ OPERATION RECOVERY INFO ---------------------------
Option ===>

Enter any of the following row commands:
L   - Browse joblog    PY   - Prompt reply Yes    PN   - Prompt reply No

Recovered job info
Application              : F100DECSCRIPT01    04/07/21  09.00
Operation                : F100 10            UNX23376

Recovery Option          : Rerun
Recovery Action          : Prompt / Job

Recovery job info
Recovery jobid           : UNX24098     Duration: 00.00.01
Work Station Name        : F100         Status  : Completed
Recovery job start/end   : 04/07/21 22.46  /  04/07/21 22.46

Recovery prompt info
Prompt Id : 3134                          Status  : Prompt reply YES
Message   : CREATE DEMO.SH ON FTA?
```

Figure 15-46 Recovery Info after the Recovery job has been executed.

The recovery job has been executed successfully and the Recovery Option (Figure 15-45) was rerun, so the failing job (F100DJ02) will be rerun and will complete successfully.

Finally, the job log is browsed for the completed F100DJ02 job (Example 15-25). The job log shows that the user is `twstest` (= USER) and that the twshome directory is `/tivoli/tws/twstest/tws` (part of the = JCLFILE line).

Example 15-25 The job log for the second run of F100DJ02 (after the RECOVERY job)

```
=============================================================
= JOB       : OPCMASTER#BB8D04BFE71A3901.J_010_F100DECSCRIPT01
= USER      : twstest
= JCLFILE   : rm /tivoli/tws/twstest/tws/demo.sh
= Job Number: 24100
= Wed 07/21/04 22:46:33 DFT
=============================================================
TWS for UNIX/JOBMANRC 8.2
AWSBJA001I Licensed Materials Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998,2003
US Government User Restricted Rights
Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
AWSBIS307I Starting /tivoli/tws/twstest/tws/jobmanrc rm
TWS for UNIX (AIX)/JOBINFO 8.2 (9.5)
Licensed Materials  Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998,2001
US Government User Restricted Rights
Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
Installed for user ''.
Locale LANG set to "C"
Now we are running the script rm /tivoli/tws/twstest/tws/demo.sh
=============================================================
= Exit Status         : 0
= System Time (Seconds) : 0     Elapsed Time (Minutes) : 0
= User Time (Seconds)   : 0
= Wed 07/21/04 22:46:33 DFT
=============================================================
```

Compare the job log output with the non-centralized script definition in Example 15-22 on page 464: The user and the twshome directory were defined as Tivoli Workload Scheduler for z/OS JCL variables (&TWSHOME and %TWSUSER). These variables have been substituted with values from the Tivoli Workload Scheduler for z/OS variable table E2EVAR (specified in the VARSUB TABLES() parameter). This variable substitution is performed when the job definition is added to the Symphony file either during normal Tivoli Workload Scheduler for z/OS plan extension or replan or if user ad hoc adds the job stream to the plan in Tivoli Workload Scheduler for z/OS.

This completes the test of the non-centralized script.

15.5.3 Verification of centralized script with JOBREC parameters

We did a verification with a job with centralized script combined with a JOBREC statement in the script library (EQQSCLIB).

The verification uses a job named F100CJ02 and centralized script, as shown in Example 15-26. The centralized script is defined in the Tivoli Workload Scheduler for z/OS JOBLIB.

Example 15-26 Centralized script for test in combination with JOBREC

```
EDIT       TWS.V8R20.JOBLIB(F100CJ02) - 01.07            Columns 00001 00072
 Command ===>                                              Scroll ===> CSR
 ****** ***************************** Top of Data ******************************
 000001 //*%OPC SCAN
 000002 //* OPC Here is an OPC JCL Variable OYMD1: &OYMD1.
 000003 //* OPC
 000004 //*%OPC RECOVER JOBCODE=(12),ADDAPPL=(F100CENTRECAPPL),RESTART=(NO)
 000005 //* OPC
 000006 echo 'Todays OPC date is: &OYMD1'
 000007 echo 'Unix system date is: '
 000008 date
 000009 echo 'OPC schedule time is: ' &CHHMMSSX
 000010 exit 12
 ****** **************************** Bottom of Data ****************************
```

The JOBREC statement for the F100CJ02 job is defined in the Tivoli Workload Scheduler for z/OS scriptlib (EQQSCLIB); see Example 15-27. It is important that the member name for the job (F100CJ02 in our example) is the same in JOBLIB and SCRPTLIB.

Example 15-27 JOBREC definition for the F100CJ02 job

```
EDIT       TWS.V8R20.SCRPTLIB(F100CJ02) - 01.07          Columns 00001 00072
Command ===>                                               Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 JOBREC
000002        RCCONDSUC('RC<7')
000003        JOBUSR(maestro)
****** **************************** Bottom of Data ****************************
```

The first time the job is run, it abends with return code 12 (due to the exit 12 line in the centralized script).

Example 15-28 shows the job log. Note the = JCLFILE line: Here you can see TWSRCMAP: RC<7, which is added because we specified RCCONDSUC('RC<7') in the JOBREC definition for the F100CJ02 job.

Example 15-28 Job log for the F100CJ02 job (ends with return code 12)

```
============================================================
= JOB       : OPCMASTER#BB8D0F9DEE6AE7C5.J_020_F100CENTSCRIPT01
= USER      : maestro
= JCLFILE   : /tivoli/tws/twstest/tws/centralized/OPCMASTER.BB8D0F9DEE6AE7C5.J_0
20_F100CENTSCRIPT01.sh TWSRCMAP: RC<7
= Job Number: 56624
= Wed 07/21/04 23:07:16 DFT
============================================================
TWS for UNIX/JOBMANRC 8.2
AWSBJA001I Licensed Materials Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998,2003
US Government User Restricted Rights
Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
AWSBIS307I Starting /tivoli/tws/twstest/tws/jobmanrc /tivoli/tws/twstest/tws/cen
tralized/OPCMASTER.BB8D0F9DEE6AE7C5.J_020_F100CENTSCRIPT01.sh
TWS for UNIX (AIX)/JOBINFO 8.2 (9.5)
Licensed Materials  Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998,2001
US Government User Restricted Rights
Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
Installed for user ''.
Locale LANG set to "C"
Todays OPC date is: 040721
Unix system date is:
Wed Jul 21 23:07:17 DFT 2004
OPC schedule time is:  23021516
============================================================
= Exit Status          : 12
= System Time (Seconds) : 0     Elapsed Time (Minutes) : 0
= User Time (Seconds)   : 0
= Wed 07/21/04 23:07:17 DFT
============================================================
```

The job log also shows that the user is set to maestro (the = USER line). This is because we specified JOBUSR(maestro) in the JOBREC statement.

Next, before the job is rerun, the JCL (the centralized script) is edited, and the last line is changed from exit 12 to exit 6. Example 15-29 shows the edited JCL.

Example 15-29 The script (JCL) for the F100CJ02 job is edited exit changed to 6

```
****** **************************** Top of Data ******************************
000001 //*>OPC SCAN
000002 //* OPC Here is an OPC JCL Variable OYMD1: 040721
000003 //* OPC
000004 //*>OPC RECOVER JOBCODE=(12),ADDAPPL=(F100CENTRECAPPL),RESTART=(NO)
000005 //* OPC MSG:
000006 //* OPC MSG: I  ***  R E C O V E R Y   A C T I O N S   T A K E N  ***
000007 //* OPC
000008 echo 'Todays OPC date is: 040721'
000009 echo
000010 echo 'Unix system date is: '
000011 date
000012 echo
000013 echo 'OPC schedule time is: ' 23021516
000014 echo
000015 exit 6
****** **************************** Bottom of Data ****************************
```

Note that the line with Tivoli Workload Scheduler for z/OS Automatic Recover has changed: The % sign has been replaced by the > sign. This means that Tivoli Workload Scheduler for z/OS has performed the recovery action by adding the F100CENTRECAPPL job stream (application).

The result after the edit and rerun of the job is that the job completes successfully. (It is marked as completed with return code = 0 in Tivoli Workload Scheduler for z/OS). The RCCONDSUC() parameter in the scriptlib definition for the F100CJ02 job sets the job to successful even though the exit code from the script was 6 (Example 15-30).

Example 15-30 Job log for the F100CJ02 job with script exit code = 6

```
============================================================
= JOB       : OPCMASTER#BB8D0F9DEE6AE7C5.J_020_F100CENTSCRIPT01
= USER      : maestro
= JCLFILE   : /tivoli/tws/twstest/tws/centralized/OPCMASTER.BB8D0F9DEE6AE7C5.J_0
20_F100CENTSCRIPT01.sh TWSRCMAP: RC<7
= Job Number: 41410
= Wed 07/21/04 23:35:48 DFT
============================================================
TWS for UNIX/JOBMANRC 8.2
AWSBJA001I Licensed Materials Property of IBM
```

```
5698-WKB
(C) Copyright IBM Corp 1998,2003
US Government User Restricted Rights
Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
AWSBIS307I Starting /tivoli/tws/twstest/tws/jobmanrc /tivoli/tws/twstest/tws/cen
tralized/OPCMASTER.BB8D0F9DEE6AE7C5.J_020_F100CENTSCRIPT01.sh
TWS for UNIX (AIX)/JOBINFO 8.2 (9.5)
Licensed Materials  Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998,2001
US Government User Restricted Rights
Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
Installed for user ''.
Locale LANG set to "C"
Todays OPC date is: 040721
Unix system date is:
Wed Jul 21 23:35:49 DFT 2004
OPC schedule time is:  23021516
============================================================
= Exit Status           : 6
= System Time (Seconds) : 0     Elapsed Time (Minutes) : 0
= User Time (Seconds)   : 0
= Wed 07/21/04 23:35:49 DFT
============================================================
```

This completes verification of centralized script combined with JOBREC statements.

15.6 Tivoli Workload Scheduler for z/OS E2E poster

As part of this IBM Redbook project, we created an Tivoli Workload Scheduler for z/OS E2E poster (authored by Michael Lowry) to help you configure the Tivoli Workload Scheduler for z/OS end-to-end scheduling environment. The poster shows all parameters that have to match in such an environment. Due to page size limitations, we had to resize the poster to fit in a book page, but you can download the PowerPoint® version from the ITSO Web site. Appendix C, “Additional material” on page 679 has instructions for downloading this file, named TWS 8.2 E2E Poster.

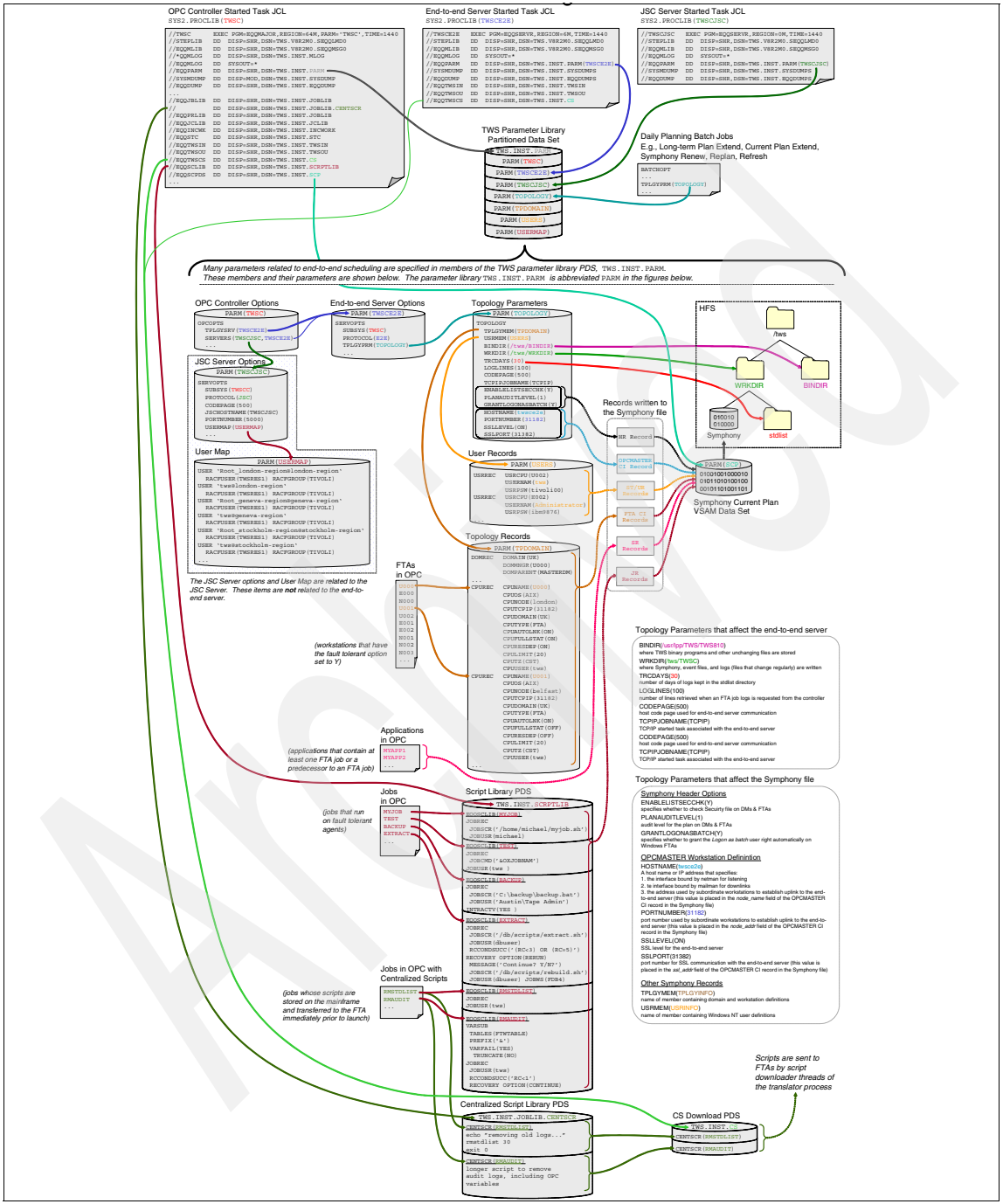


Figure 15-47 Tivoli Workload Scheduler for z/OS E2E poster

16

Using the Job Scheduling Console with Tivoli Workload Scheduler for z/OS

This chapter covers the architecture and implementation of Job Scheduling Console Version 1.4, with the following sections:

- Job Scheduling Console (JSC)
- Activating support for the Job Scheduling Console
- Installing the connectors
- Installing the Job Scheduling Console step by step
- ISPF and JSC side by side

In the final section of this chapter, we will compare using the Interactive System Productivity Facility (ISPF) screens on the mainframe for application management versus using the Job Scheduling Console.

16.1 Job Scheduling Console

The Job Scheduling Console (JSC) is a Java-based GUI for controlling and monitoring workload on the mainframe and other platforms. The first version of JSC was released at the same time as Tivoli OPC Version 2.3. The current version of JSC (1.4) has been updated with several new functions specific to Tivoli Workload Scheduler for z/OS. JSC provides a common interface to both Tivoli Workload Scheduler for z/OS and Tivoli Workload Scheduler.

16.1.1 JSC components

- Tivoli Workload Scheduler Engine/Tivoli Workload Scheduler for z/OS server
- Tivoli Management Framework
  - TMR Server of a Managed Node
- Job Scheduling Services (JSS)
- Tivoli Workload Scheduler Connector
- Tivoli Workload Scheduler for z/OS Connector
- Job Scheduling Console (JS Console)

16.1.2 Architecture and design

There are two types of connectors and they function quite differently.

The Tivoli Workload Scheduler for z/OS Connector communicates over TCP/IP with the Tivoli Workload Scheduler for z/OS Controller running on a mainframe computer.

The Tivoli Workload Scheduler Connector performs direct local reads and writes of the Tivoli Workload Scheduler plan and database files. The Tivoli Workload Scheduler Connector must be instantiated on the host where the Tivoli Workload Scheduler Engine is installed, so that it can access the plan and database files locally. A good place to install this is for the Primary Domain Manager.

The Job Scheduling Console can be run on almost any platform. Using the JSC, an operator can access both Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS scheduling engines. JSC communicates with the engines via the Job Scheduling Services (JSS) and the appropriate connector. JSS and the connectors must be installed on top of the Tivoli Management Framework (TMF) as shown in Figure 16-1 on page 483.

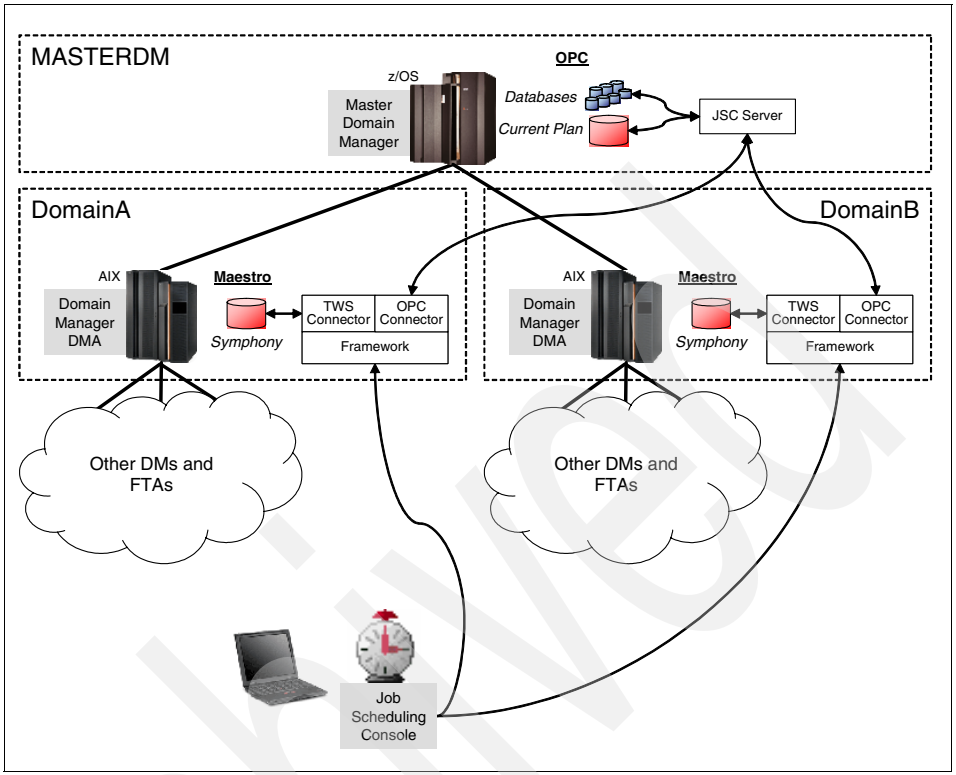


Figure 16-1 Job Scheduling Console and Tivoli Workload Scheduler Connectors

16.2 Activating support for the Job Scheduling Console

To activate support for use of the Tivoli Workload Scheduler Job Scheduling Console (JSC), perform the following steps:

1. “Install and start JSC Server” on page 484.
2. “Installing and configuring Tivoli Management Framework” on page 490.
3. Install Job Scheduling Services (JSS) in Tivoli Management Framework. The JSS is installed automatically when you specify an agent and install the connector. For more information about Tivoli Workload Scheduler for z/OS, see *IBM Tivoli Workload Schedulers Job Scheduling Console V1.4 User’s Guide*, SC32-1257.

4. To be able to work with Tivoli Workload Scheduler for z/OS (OPC) controllers from the JSC (“Creating connector instances” on page 492):
   a. Install the Tivoli Workload Scheduler for z/OS connector in Tivoli Management Framework.
   b. Create instances in Tivoli Management Framework that point to the Tivoli Workload Scheduler for z/OS controllers you want to access from the JSC.
5. To be able to work with Tivoli Workload Scheduler domain managers or fault-tolerant agents from the JSC:
   a. Install the Tivoli Workload Scheduler connector in Tivoli Management Framework. Note that the Tivoli Management Framework server or managed node must be installed on the machine where the Tivoli Workload Scheduler instance is installed.
   b. Create instances in Tivoli Management Framework that point to the Tivoli Workload Scheduler domain managers or fault-tolerant agents that you would like to access from the JSC.
6. Install the JSC on the workstations where it should be used (see “Installing the Job Scheduling Console step by step” on page 499).

The following sections describe installation steps in more detail.

16.2.1 Install and start JSC Server

To use Tivoli Workload Scheduler Job Scheduling Console for communication with Tivoli Workload Scheduler for z/OS, you must initialize the Tivoli Workload Scheduler for z/OS Connector (also called the OPC Connector). The connector forms the bridge between the Tivoli Workload Scheduler Job Scheduling Console and the Tivoli Workload Scheduler for z/OS product.

The JSC communicates with Tivoli Workload Scheduler for z/OS through the scheduler server using the TCP/IP protocol. The JSC needs the server to run as a started task in a separate address space. The Tivoli Workload Scheduler for z/OS server communicates with Tivoli Workload Scheduler for z/OS and passes the data and return codes back to the connector.

The security model that is implemented for Tivoli Workload Scheduler Job Scheduling Console is similar to that already implemented by other Tivoli products that have been ported to z/OS (namely IBM Tivoli User Administration and IBM Tivoli Security Management). The Tivoli Framework security handles the initial user verification, but it is necessary to obtain a valid corresponding RACF user ID to be able to work with the security environment in z/OS.

Even though it is possible to have one server started task handling end-to-end scheduling, JSC communication, and even APPC communication, we

recommend having a server started task dedicated to JSC communication (*SERVOPTS PROTOCOL(JSC)*). This has the advantage that you do not have to stop the whole end-to-end server process if only the JSC communication has be restarted.

We will install a server dedicated to JSC communication and call it the JSC Server.

When JSC is used to access the Tivoli Workload Scheduler for z/OS controller through the JSC Server, the JSC Server uses the Tivoli Workload Scheduler for z/OS program interface (PIF) to interface with the controller.

You can find an example of the started task procedure in installation member EQQSER in the sample library that is generated by the EQQJOBS installation aid. An example of the initialization statements can be found in the EQQSERP member in the sample library generated by the EQQJOBS installation aid. After the installation of the JSC Server, you can get almost the same functionality from the JSC as you have with the Tivoli Workload Scheduler for z/OS IPSF interface.

Configure and start the JSC Server and verify the start

First, create the started task procedure for the JSC Server. The EQQSER member in the sample library can be used. Take the following into consideration when customizing the EQQSER sample:

- Make sure that the C runtime library is concatenated in the server JCL (CEE.SCEERUN) in STEPLLIB if it is not in the LINKLIST.
- If you have multiple TCP/IP stacks or if the name of the procedure that was used to start the TCPIP address space is different from `TCPIP`, introduce the SYSTCPD DD card pointing to a data set containing the TCPIPJOBNAME parameter. (See DD SYSTCPD in the TCP/IP manuals.)
- Customize the JSC Server initialization parameters file. (See the EQQPARM DD statement in the server JCL.) The installation member EQQSERP already contains a template.

For information about the JSC Server parameters, refer to *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning,* SC32-1265.

We used the JSC Server initialization parameters shown in Example 16-1. Also see Figure 16-2 on page 487.

Example 16-1 The JSC Server initialization parameter

```
/*********************************************************************/
/* SERVOPTS: run-time options for the TWSCJSC started task           */
/*********************************************************************/
SERVOPTS   SUBSYS(TWSC)
```

```
/*-------------------------------------------------------------------*/
/* TCP/IP server is needed for JSC GUI usage. Protocol=JSC           */
/*-------------------------------------------------------------------*/
          PROTOCOL(JSC)                 /* This server is for JSC   */
          JSCHOSTNAME(TWSCJSC)          /* DNS name for JSC         */
          USERMAP(USERS)                /* RACF user / TMF adm. map */
          PORTNUMBER(38888)             /* Portnumber for JSC comm. */
          CODEPAGE(IBM-037)             /* Codep. EBCIDIC/ASCII tr. */
/*-------------------------------------------------------------------*/
/* CALENDAR parameter is mandatory for server when using TCP/IP      */
/* server.                                                           */
/*-------------------------------------------------------------------*/
INIT      ADOICHK(YES)                  /* ADOI Check ON            */
          CALENDAR(DEFAULT)             /* Use DEFAULT calendar     */
          HIGHDATE(711231)              /* Default HIGHDATE         */
```

The SUBSYS(), PROTOCOL(JSC), CALENDAR(), and HIGHDATE() are mandatory for using the Tivoli Job Scheduling Console. Make sure that the port you try to use is not reserved by another application.

If JSCHOSTNAME() is not specified, the default is to use the host name that is returned by the operating system.

Note: We got an error when trying to use the JSCHOSTNAME with a host name instead of an IP address (EQQPH18E COMMUNICATION FAILED). This problem is fixed with APAR PQ83670.

Remember that you always have to define OMVS segments for Tivoli Workload Scheduler for z/OS server started task user IDs.

Optionally, the JSC Server started task name can be defined in the Tivoli Workload Scheduler for z/OS controller OPCOPTS SERVERS() parameter to let the Controller start and stop the JSC Server task when the Controller itself is started and stopped (Figure 16-2 on page 487).

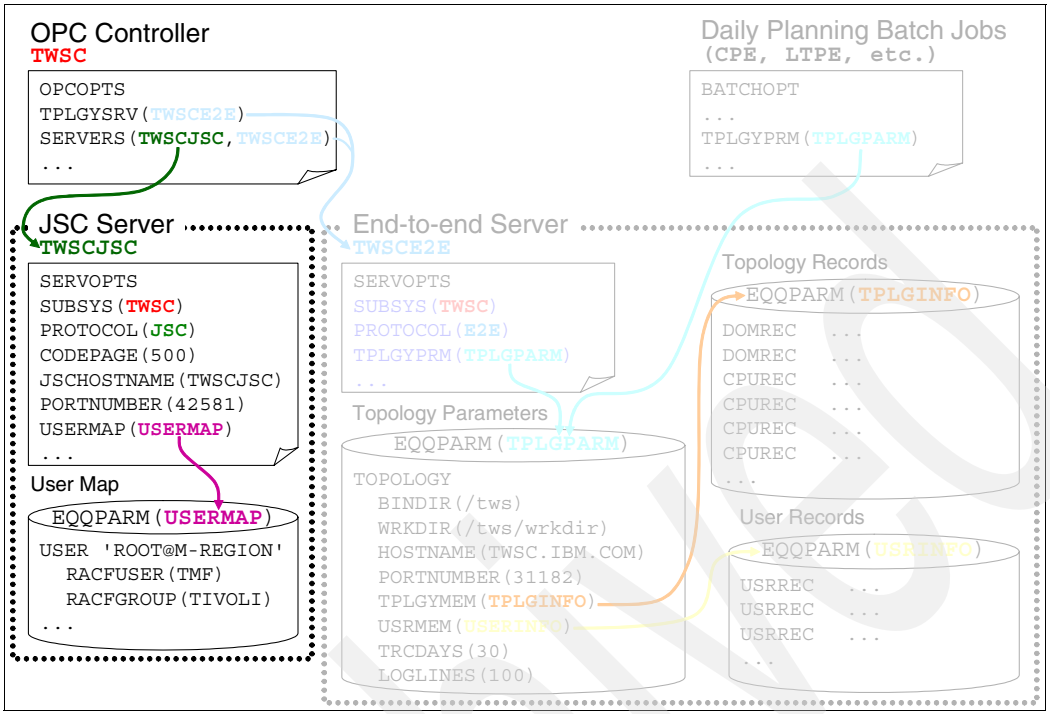


Figure 16-2 JSC Server that communicates with TWSC Controller

After the configuration and customization of the JSC Server initialization statements and the JSC Server started task procedure, we started the JSC Server and saw the messages in Example 16-2 during start.

Example 16-2 Messages in EQQMLOG for JSC Server when started

```
EQQZ005I OPC SUBTASK SERVER           IS BEING STARTED
EQQPH09I THE SERVER IS USING THE TCP/IP PROTOCOL
EQQPH28I THE TCP/IP STACK IS AVAILABLE
EQQPH37I SERVER CAN RECEIVE JSC REQUESTS
EQQPH00I SERVER TASK HAS STARTED
```

Controlling access to Tivoli Workload Scheduler for z/OS from the JSC

The Tivoli Framework performs a security check, verifying the user ID and password, when a user tries to use the Job Scheduling Console. The Tivoli Framework associates each user ID and password with an administrator. Tivoli Workload Scheduler for z/OS resources are protected by RACF.

The JSC user should have to enter only a single user ID and password combination, not at both the Tivoli Framework level and then again at the Tivoli Workload Scheduler for z/OS level.

The security model is based on having the Tivoli Framework security handle the initial user verification while obtaining a valid corresponding RACF user ID. This makes it possible for the user to work with the security environment in z/OS. The z/OS security is based on a table mapping the Tivoli Framework administrator to an RACF user ID. When a Tivoli Framework user tries to initiate an action on z/OS, the Tivoli administrator ID is used as a key to obtain the corresponding RACF user ID.

The JSC Server uses the RACF user ID to build the RACF environment to access Tivoli Workload Scheduler for z/OS services, so the Tivoli Administrator must relate, or map, to a corresponding RACF user ID.

There are two ways of getting the RACF user ID:

- Use the RACF Tivoli-supplied predefined resource class, TMEADMIN.

  Consult the section about implementing security in Tivoli Workload Scheduler for z/OS in *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning,* SC32-1265, for the complete setup of the TMEADMIN RACF class.

- Use a new OPC Server Initialization Parameter to define a member in the file identified by the EQQPARM DD statement in the server startup job.

  This member contains all of the associations for a TME user with an RACF user ID. You should set the parameter USERMAP in the JSC Server SERVOPTS Initialization Parameter to define the member name.

Use of the USERMAP(USERS)

We used the JSC Server SERVOPTS USERMAP(USERS) parameter to define the mapping between Tivoli Framework Administrators and z/OS RACF users.

USERMAP(USERS) means that the definitions (mappings) are defined in a member named USERS in the EQQPARM library. See Figure 16-3 on page 489.

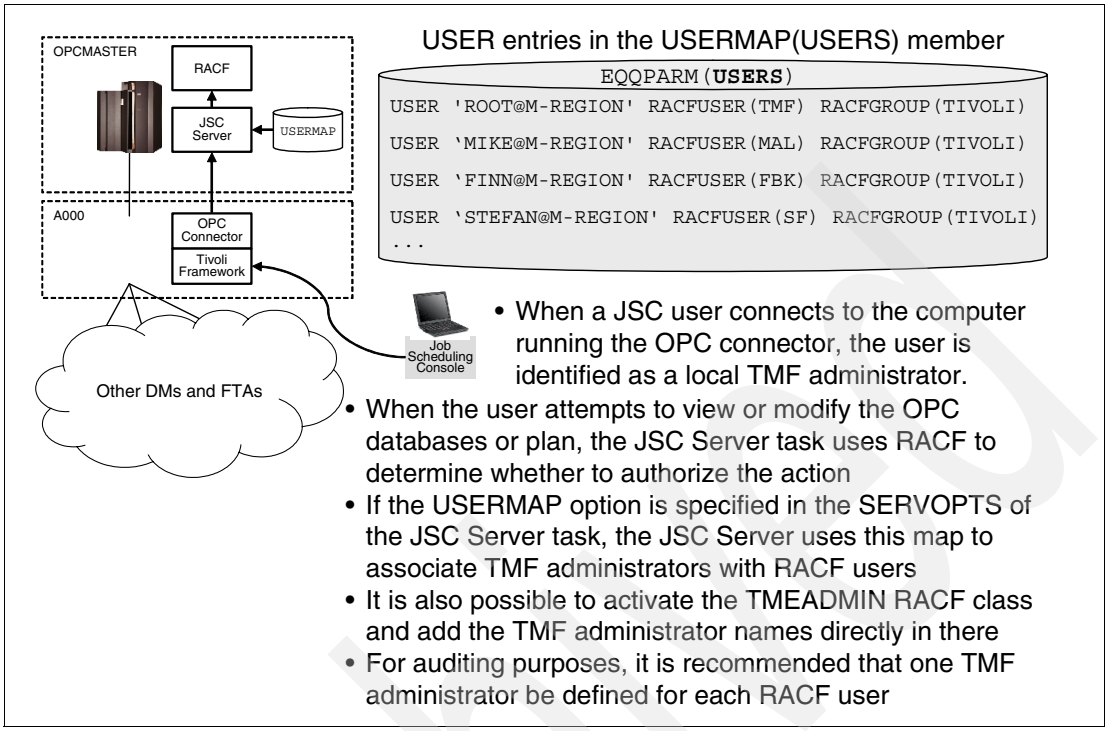


Figure 16-3 Relationship between TMF administrators and RACF users via USERMAP

For example, in the definitions in the USERS member in EQQPARM in Figure 16-3, TMF administrator `MIKE@M-REGION` is mapped to RACF user `MAL` (MAL is member of RACF group TIVOLI). If MIKE logs in to the TMF with name M-REGION and MIKE@M-REGION works with the Tivoli Workload Scheduler for z/OS Controller from the JSC, he will have the access defined for RACF user MAL. In other words, the USER definition maps TMF Administrator MIKE@M-REGION to RACF user MAL.

Whatever MIKE@M-REGION is doing from the JSC in the Controller will be performed with the RACF authorization defined for the MAL user. All logging in RACF will also be done for the MAL user.

The TMF Administrator is defined in TMF with a certain authorization level. The TMF Administrator must have the USER role to be able to use the Tivoli Workload Scheduler for z/OS connector.

Notes:

- If you decide to use the USERMAP to map TMF administrators to RACF users, you should be aware that users with update access to the member with the mapping definitions (the USERS member in our example) can get access to the Tivoli Workload Scheduler for z/OS Controller by editing the mapping definitions.

  To avoid any misuse, make sure that the member with the mapping definitions is protected according to your security standards. Or use the standard RACF TMEADMIN resource class in RACF to do the mapping.
- To be able to audit what different JSC users do in Tivoli Workload Scheduler for z/OS, we recommend that you establish a one-to-one relationship between the TMF Administrator and the corresponding RACF user. (That is, you should not allow multiple users to use the same TMF Administrator by adding several different logons to one TMF Administrator.)

For information about troubleshooting the OPC Connector, refer to 18.7, “OPC Connector troubleshooting” on page 645.

16.2.2 Installing and configuring Tivoli Management Framework

As we discussed, for the new Job Scheduling Console interface to communicate with the scheduling engines, a few other components must be installed.

When installing Tivoli Workload Scheduler 8.2.1 using the ISMP installer GUI, you are given the option to install the Tivoli Workload Scheduler connector. If you choose this option, the installer program automatically installs the following components:

- Tivoli Management Framework 4.1.1, configured as a TMR server
- Job Scheduling Services 1.2
- Tivoli Workload Scheduler Connector 8.2.1

The Tivoli Workload Scheduler 8.2.1 installer GUI will also automatically create a Tivoli Workload Scheduler connector instance and a TMF administrator associated with your Tivoli Workload Scheduler user. Letting the installer do the work of installing and configuring these components is generally a good idea because it saves the trouble of performing each of these steps individually.

If you choose not to let the Tivoli Workload Scheduler 8.2.1 installer install and configure these components for you, you can install them later. The following instructions for getting a TMR server installed and up and running, and to get Job Scheduling Services and the connectors installed, are primarily intended for

environments that do not already have a TMR server, or one in which a separate TMR server will be installed for IBM Tivoli Workload Scheduler.

In 16.3.1, “Creating connector instances” on page 492 and 16.3.2, “Creating TMF administrators for Tivoli Workload Scheduler” on page 495, we discuss in more detail the steps specific to end-to-end scheduling: creating connector instances and TMF administrators.

The Tivoli Management Framework is easy to install. If you already have the Framework installed in your organization, it is not necessary to install the components specific to Tivoli Workload Scheduler (the JSS and connectors) on a node in your existing Tivoli Managed Region. You may prefer to install a stand-alone TMR server solely for the purpose of providing the connection between the IBM Tivoli Workload Scheduler suite and its interface, the JSC. If your existing TMR is busy with other operations, such as monitoring or software distribution, you might want to consider installing a separate stand-alone TMR server for Tivoli Workload Scheduler.

16.2.3 Install Job Scheduling Services

Follow the instructions in *16.4, “Installing the Job Scheduling Console step by step” on page 499* to install JSS. JSS is simply a library used by the Framework, and it is a prerequisite of the connectors.

The hardware and software prerequisites for the Job Scheduling Services are:

- Software

  IBM Tivoli Management Framework: Version 3.7.1 or later for Microsoft Windows, AIX, HP-UX, Sun Solaris, and Red Hat and SUSE Linux.

- Hardware
  - CD-ROM drive for installation
  - Approximately 4 MB of free disk space

For the latest information about the supported versions of these platforms, refer to *IBM Tivoli Workload Scheduler Job Scheduling Console Release Notes, Feature level 1.4*, SC32-1258.

16.3 Installing the connectors

We recommend that when installing the Tivoli Workload Scheduler connector, you do not select the **Create Instance** check box. Create the instances after the connector has been installed.

The hardware and software prerequisites for the Tivoli Workload Scheduler for z/OS connector are:

- Software:
  - IBM Tivoli Management Framework: Version 3.7.1 or later
  - Tivoli Workload Scheduler for z/OS 8.1, or Tivoli OPC 2.1 or later
  - Tivoli Job Scheduling Services 1.2
  - TCP/IP network communications
  - A Tivoli Workload Scheduler for z/OS user account (required), which you can create beforehand or have the setup program create for you
- Hardware:
  - CD-ROM drive for installation.
  - Approximately 3 MB of free disk space for the installation. In addition, the Tivoli Workload Scheduler for z/OS connector produces log files and temporary files, which are placed on the local hard drive.

Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS connector are supported on the following platforms: Microsoft Windows, IBM AIX, HP-UX, Sun Solaris, and Red Hat and SUSE Linux. For the latest information about the supported versions of these platforms, refer to *IBM Tivoli Workload Scheduler Job Scheduling Console Release Notes,* Feature level 1.4, SC32-1258.

16.3.1 Creating connector instances

The connectors tell the Framework how to communicate with the different types of scheduling engine.

To control the workload of the entire end-to-end scheduling network from the Tivoli Workload Scheduler for z/OS Controller, it is necessary to create a Tivoli Workload Scheduler for z/OS connector instance to connect to that Controller.

It may also be a good idea to create a Tivoli Workload Scheduler connector instance on a fault-tolerant agent or domain manager. Sometimes the status may get out of sync between an FTA or DM and the Tivoli Workload Scheduler for z/OS Controller. When this happens, it is helpful to be able to connect directly to that agent and get the status directly from there. Retrieving job logs (standard lists) is also much faster through a direct connection to the FTA than through the Tivoli Workload Scheduler for z/OS Controller.

Creating a Tivoli Workload Scheduler for z/OS Connector instance

You have to create at least one Tivoli Workload Scheduler for z/OS connector instance for each z/OS Controller that you want to access with the Tivoli Job Scheduling Console. This is done using the **wopcconn** command.

In our test environment, we wanted to be able to connect to a Tivoli Workload Scheduler for z/OS Controller running on a mainframe with the host name twscjsc. On the mainframe, the Tivoli Workload Scheduler for z/OS TCP/IP server listens on TCP port 3011. ITSO05 is the name of the TMR-managed node where we created the connector instance. We called the new connector instance twsc. Here is the command we used:

```
wopcconn -create -h rome -e TWSC -a twscjsc -p 3011
```

In this command we are creating an instance that will enable the connector to connect to the TCP/IP server. The `-h` followed by `rome` is the name or the ID of the managed node where we are creating the instance. The `-e` or connector_name is the name of the new instance (TWSC). The `-a` or address is for the IP address of the z/OS system where the scheduler subsystem to which you want to connect is installed. The `-p` is for the port number as described above on the TCP/IP server so the connector can connect to it.

The result of this will be that when we use JSC to connect to ITSO05, a new connector instance called TWSC appears in the Job Scheduling list on the left side of the window. We can access the Tivoli Workload Scheduler for z/OS scheduling engine by clicking that new entry in the list.

It is also possible to run **wopcconn** in interactive mode. To do this, just run **wopcconn** with no arguments as the following example.

Example 16-3 wopcconn command

```
[root@rome][/Tivoli]-> wopcconn


IBM Tivoli Workload Scheduler for z/OS Connector manage program
                Main menu

1. Create new IBM Tivoli Workload Scheduler for z/OS Connector instance
2. Manage an existing IBM Tivoli Workload Scheduler for z/OS Connector
instance

0. Exit
```

Creating a Tivoli Workload Scheduler Connector instance

Remember that a Tivoli Workload Scheduler connector instance must have local access to the Tivoli Workload Scheduler engine with which it is associated. This is done using the **wtwsconn.sh** command.

In our test environment, we wanted to be able to use JSC to connect to a Tivoli Workload Scheduler engine on the host rome. rome has two Tivoli Workload Scheduler engines installed, so we had to be sure to specify that the path of the Tivoli Workload Scheduler engine we specify when creating the connector is the path to the right Tivoli Workload Scheduler engine. We called the new connector instance Rome-A to reflect that this connector instance would be associated with the RomeA engine on this host (as opposed to the other Tivoli Workload Scheduler engine, TWS-B). Here is the command we used:

```
wtwsconn.sh -create -h rome -n Rome-A -t /tivoli/TWS/tws-a
```

The result is that when we use JSC to connect to ITSO05, a new connector instance called TWS-A appears in the Job Scheduling list on the left side of the window. We can access the TWS-A scheduling engine by clicking that new entry in the list.

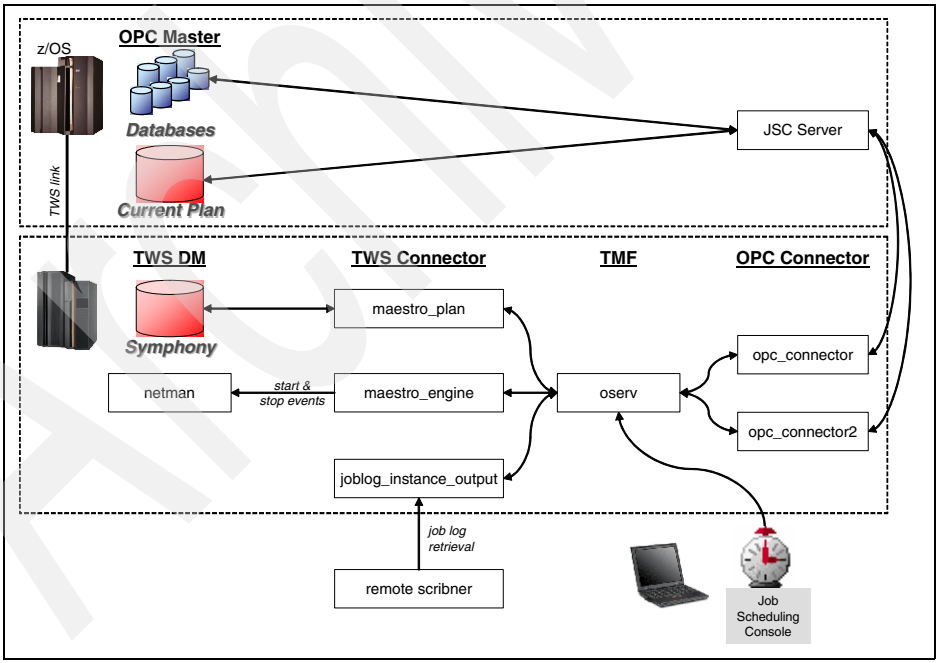


Figure 16-4 IBM Tivoli Workload Scheduler Connectors

Figure 3-3 shows the architecture of the Tivoli Workload Scheduler Connectors. A Tivoli Workload Scheduler Connector consists of five different programs, and each program has a specific function. The oserv program calls the connector program that is appropriate for the action being performed in the JSC (Job Scheduling Console):

1. The maestro_database performs direct reads and writes of the Tivoli Workload Scheduler database files in TWSHOME/mozart (just like composer).
2. The maestro_plan reads the Symphony file directly but it submits changes to the Symphony file by queuing events to the Mailbox.msg file (just as conman does).
3. The maestro_engine submits start and stop events to netman via the NetReq.msg file (just as conman does).
4. The maestro_x_server provides extra SAP R/3-specific functions to the JSC. It calls r3batch to retrieve information from and make changes to a remote SAP system; this is what enables the JSC to provide R/3-specific options when creating jobs on SAP R/3 extended agents.
5. The joblog_retriever retrieves the requested job log from scribner on the FTA where the job ran by contacting scribner on the remote host (just as conman does).

Tivoli Workload Scheduler Connector programs reside in $BINDIR/Maestro/bin.

16.3.2 Creating TMF administrators for Tivoli Workload Scheduler

When a user logs on to the Job Scheduling Console, the Tivoli Management Framework verifies that the user's logon is listed in an existing TMF administrator.

TMF administrators for Tivoli Workload Scheduler

A Tivoli Management Framework administrator must be created for the Tivoli Workload Scheduler user. Additional TMF administrators can be created for other users who will access Tivoli Workload Scheduler using JSC.

TMF administrators for Tivoli Workload Scheduler for z/OS

The Tivoli Workload Scheduler for z/OS TCP/IP server associates the Tivoli administrator to an RACF user. If you want to be able to identify each user uniquely, one Tivoli Administrator should be defined for each RACF user. If operating system users corresponding to the RACF users do not already exist on the TMR server or on a managed node in the TMR, you must first create one OS user for each Tivoli administrator that will be defined. These users can be created on the TMR server of any managed node in the TMR. After you have created those users, you can simply add those users' logins to the TMF administrators that you create.

Important: When creating users or setting their passwords, disable any option that requires the user to set a password at the first logon. If the operating system requires the user's password to change at the first logon, the user will have to do this before being able to log on via the Job Scheduling Console.

Creating TMF administrators

If Tivoli Workload Scheduler 8.2 is installed using the graphical ISMP installer, you have the option of installing the Tivoli Workload Scheduler connector automatically during Tivoli Workload Scheduler installation. If you choose this option, the installer will create one TMF administrator automatically.

We still recommend that you create one Tivoli Management Framework Administrator for each user who will use JSC.

Perform the following steps from the Tivoli desktop to create a new TMF administrator:

1. Double-click the **Administrators** icon and select **Create** → **Administrator,** as shown in Figure 16-5.

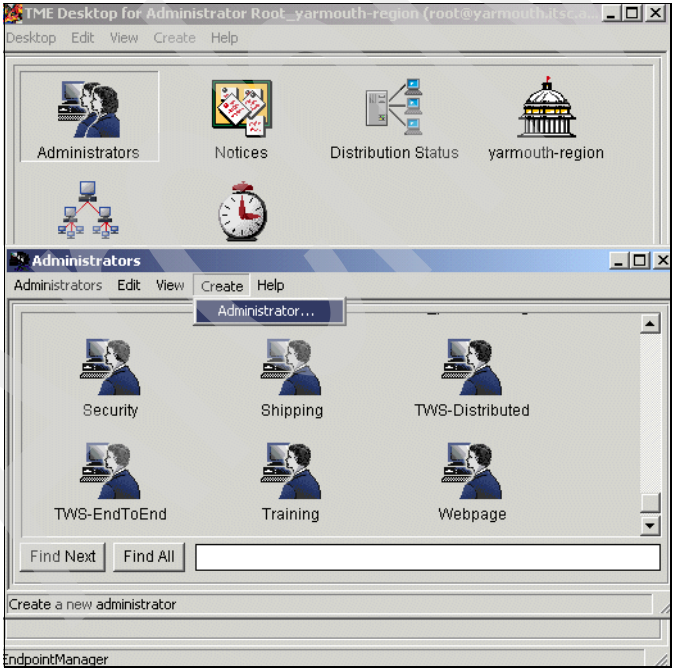


Figure 16-5 Create Administrator window

2. Enter the Tivoli Administrator name you want to create.

3. Click **Set Logins** to specify the login name (Figure 16-6). This field is important because it is used to determine the UID with which many operations are performed and represents a UID at the operating system level.

Figure 16-6 Create Administrator - specify login name

4. Type in the login name and press Enter. Click **Set & Close** (Figure 16-7).

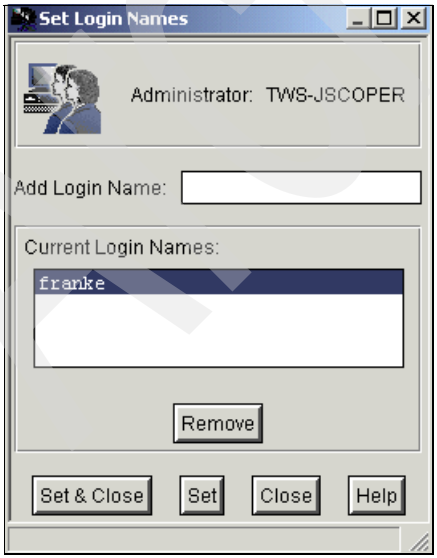


Figure 16-7 Set Login Names

5. Enter the name of the group. This field is used to determine the GID under which many operations are performed. Click **Set & Close**.

The TMR roles you assign to the administrator depend on the actions the user will need to perform.

Table 16-1 Authorization roles required for connector actions

An Administrator with this role...	Can perform these actions
User	Use the instance View instance settings
Admin, senior, or super	Use the instance View instance settings Create and remove instances Change instance settings Start and stop instances

6. Click the **Set TMR Roles** icon and add the desired role or roles (Figure 16-8).

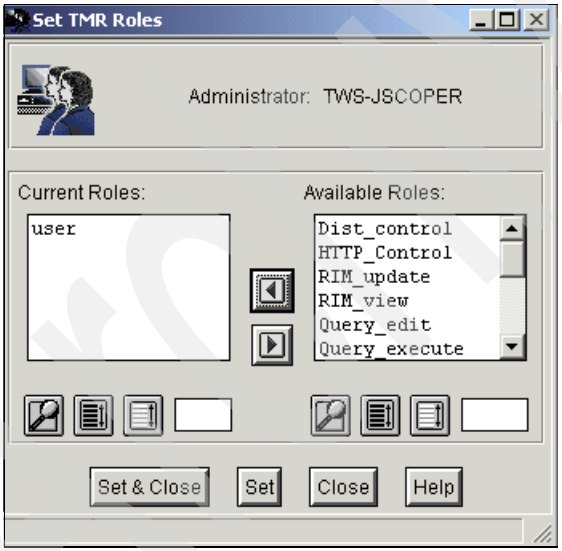


Figure 16-8 Set TMR roles

7. Click **Set & Close** to finish your input. This returns you to the Administrators desktop (Figure 16-9 on page 499).

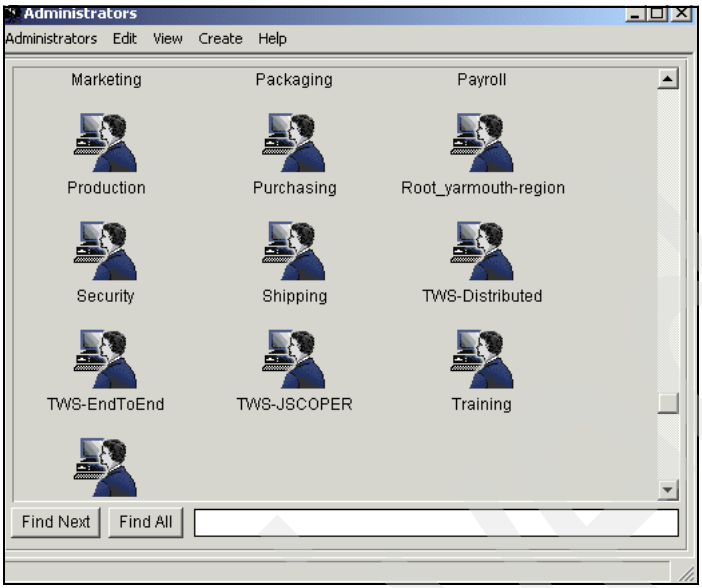


Figure 16-9 Tivoli Administrator desktop

16.4 Installing the Job Scheduling Console step by step

Tivoli Workload Scheduler for z/OS is shipped with the latest version (Version 1.4) of the Job Scheduling Console. We recommend that you use this version because it contains the best functionality and stability. For the latest information about supported platforms and versions for Job Scheduling Console V1.4, refer to **Workload Scheduler** → **Release Notes** → **Engine** at:

http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?toc=/com.ibm.tivoli.itws.doc/toc.xml

Hardware and software prerequisites

For Job Scheduling Console hardware and software prerequisites, see **Workload Scheduler** → **Release Notes** → **Prerequisites** at:

http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?toc=/com.ibm.tivoli.itws.doc/toc.xml

Important: You should use the same versions of the scheduler and the connector.

The Tivoli Workload Scheduler for z/OS connector can support any Operations Planning and Control V2 release level as well as Tivoli Workload Scheduler for z/OS 8.2.

Installing the JSC is a very simple process:

1. Insert the Tivoli Job Scheduling Console CD-ROM into the system CD-ROM drive or mount the CD-ROM from a drive on a remote system.
   - On Microsoft Windows:

     setup.exe
   - On UNIX:

     setup.bin
2. The first step in the installation process is language selection (Figure 16-10). Choose from English (default), French, German, Italian, Japanese, Korean, Portuguese, Simplified Chinese, Spanish and Traditional Chinese.

Figure 16-10 JSC Language selection

3. At the Welcome window (Figure 16-11) you can repair an existing installation, perform a new installation, or upgrade from a prior version of the JSC.

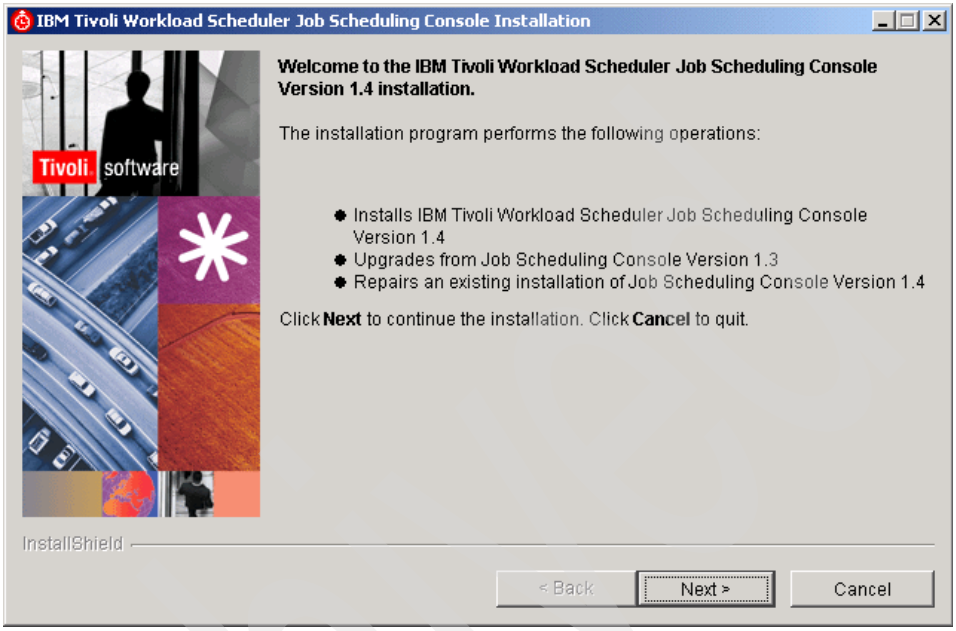


Figure 16-11 Welcome Installation window for JSC Version 1.4

4. Accept the License Agreement and click **Next** (Figure 16-12).

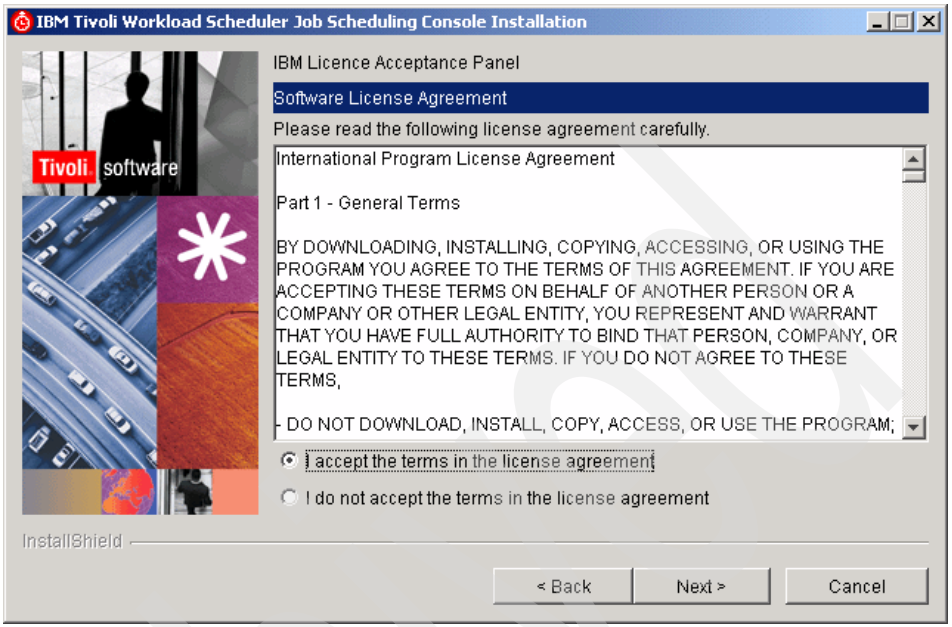


Figure 16-12 JSC Version 1.4 License Agreement

5. Select the destination directory or choose your own directory (Figure 16-13).

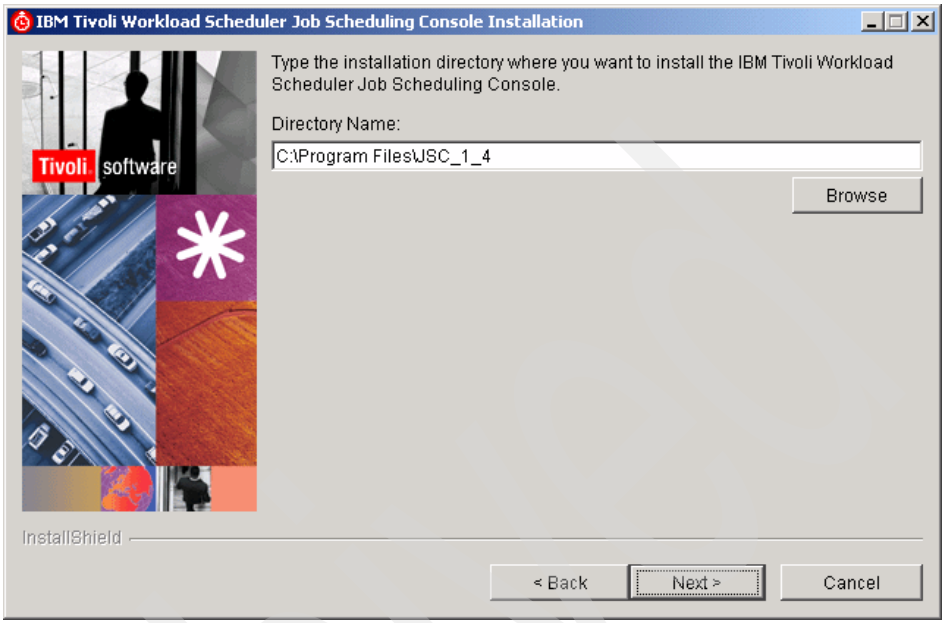


Figure 16-13 JSC Destination Directory

6. Choose whether to have an icon created for JSC Version 1.4 (Figure 16-14).

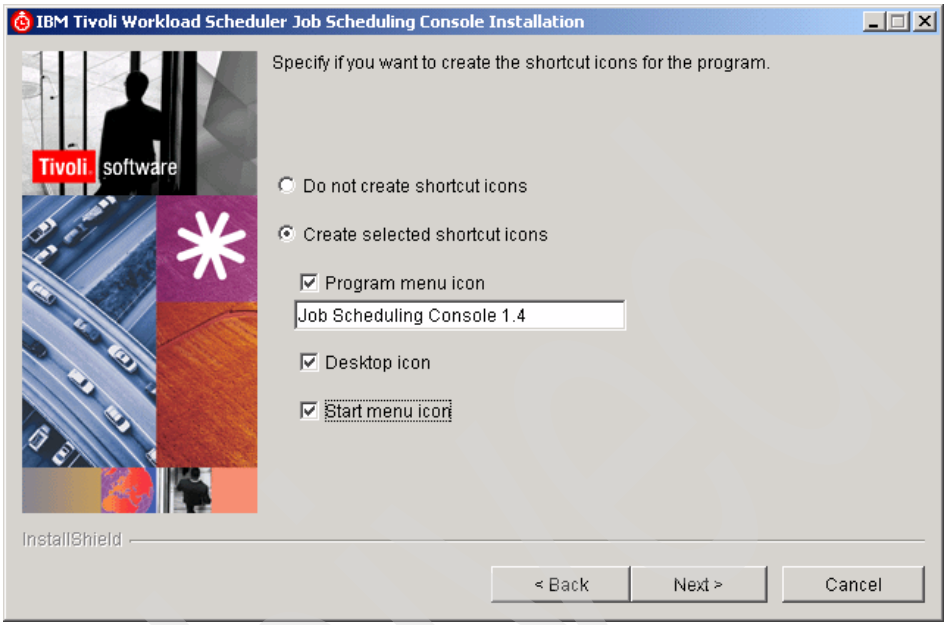


Figure 16-14 JSC icon

7. Confirm the installation directory of JSC Version 1.4 and indicate the amount of space that will used for the install (Figure 16-15) and click **Next**.

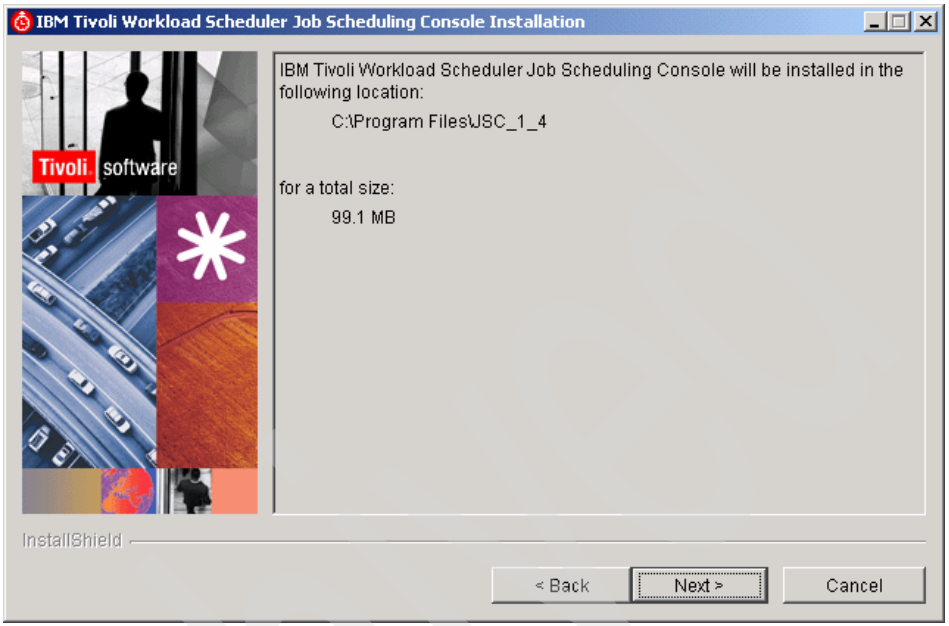


Figure 16-15 Installation confirmation of JSC 1.4

8. Installation begins. When this is complete, you will see the Finish window (Figure 16-16), which indicates that the JSC was installed successfully.

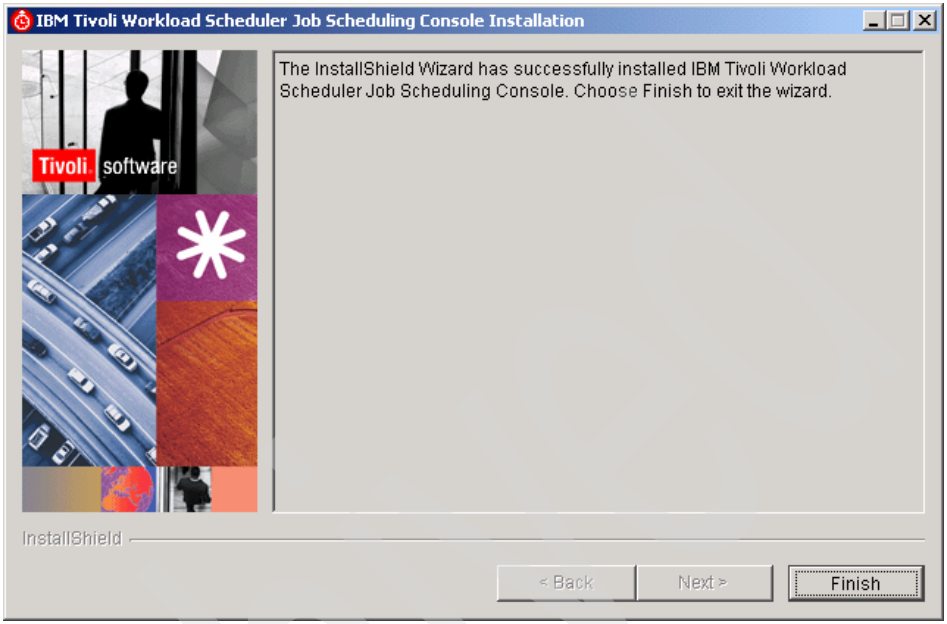


Figure 16-16 JSC successfully installed

Starting the Job Scheduling Console

Use the following steps to start the JSC, depending on your platform:

- On Windows:

  Depending on the shortcut location that you specified during installation, click the **JS Console** icon or select the corresponding item in the Start menu.

- On Windows 95 and Windows 98:

  You can also start the JSC from the command line. Type `runcon` from the \bin\java subdirectory of the installation path.

- On AIX:

  Type `./AIXconsole.sh`

- On Sun Solaris:

  Type `./SUNconsole.sh`

A Job Scheduling Console start-up window is displayed (Figure 16-17).

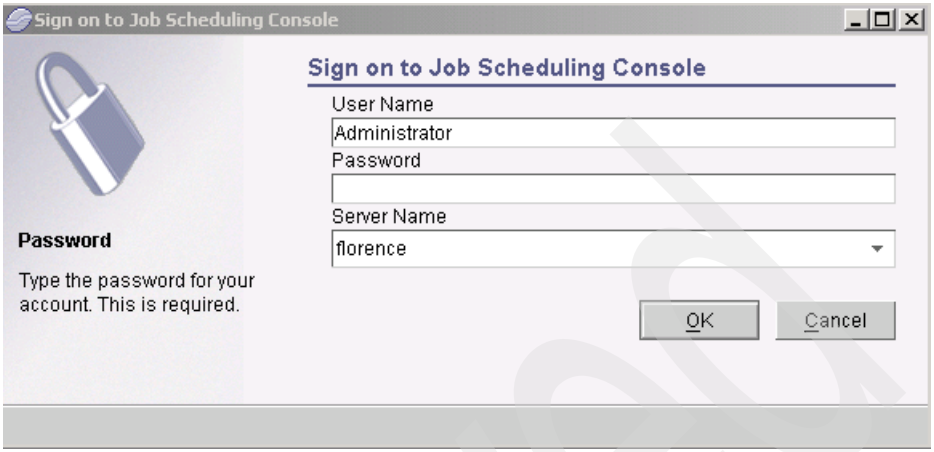


Figure 16-17 JSC login window

Enter the following information then click the **OK** button to proceed:

User name The user name of the person who has permission to use the Tivoli Workload Scheduler for z/OS Connector instances

Password The password for the Tivoli Framework administrator

Host Machine The name of the Tivoli-managed node that runs the Tivoli Workload Scheduler for z/OS Connector

16.5 ISPF and JSC side by side

In this section, we compare using the Interactive System Productivity Facility (ISPF) screens on the mainframe for application management versus using the Job Scheduling Console. Either may be used for managing applications in a Tivoli Workload Scheduler for z/OS end-to-end environment.

The ISPF dialog is installed along with the Tivoli Workload Scheduler for z/OS Controller. As we have seen in the previous sections, the JSC is installed on a Windows or UNIX system and uses the Tivoli Workload Scheduler for z/OS Connector (also called the OPC Connector) to communicate with the Controller on the mainframe.

The reasons for choosing whether to use the ISPF panels or the JSC screens are varied. Care should be taken to utilize the method that best suits the environment and the employees using Tivoli Workload Scheduler for z/OS. The ISPF panels and the JSC screens can be engaged simultaneously in the same

environment, and most customers will use a combination of the two. Because of limitations such as editing JCL, the ISPF panels should remain available to some members of the scheduling team. In an environment that has many schedulers that are UNIX- or Windows-oriented, an environment with a majority of users on the JSC may be appropriate.

16.5.1 Starting applications management

First we compare ISPF and JSC views for starting applications management.

The TWS ISPF panels

The display of all TWS ISPF panel names is toggled on and off by entering the command PANELID on the command line. We will use PANELID to show the names of the panels used in this chapter. The entry panel to the ISPF dialogs is EQQOPCAP. It shows the current version of Tivoli Workload Scheduler for z/OS and the name of the controller that is managing Tivoli Workload Scheduler for z/OS. Figure 16-18 shows the EQQOPCAP panel for Tivoli Workload Scheduler for z/OS.

```
EQQOPCAP ------------ Tivoli Workload Scheduler for z/OS  ----------------------

Welcome to TWS V820. You are communicating with 082C

Select one of the following options and press ENTER.

0 OPTIONS            - Define OPC dialog user parameters and options

1 DATABASE           - Display or update OPC data base information
2 LTP                - Long Term Plan query and update
3 DAILY PLANNING     - Produce daily plans, real and trial
4 WORK STATIONS      - Work station communication
5 MCP                - Modify the Current Plan
6 QCP                - Query the status of work in progress
7 OLD OPERATIONS     - Restart old operations from the DB2 repository

9 SERVICE FUNC       - Perform OPC service functions
10 OPTIONAL FUNC     - Optional functions
X EXIT               - Exit from the OPC dialog









Option ===> _
```

Figure 16-18 The EQQOPCAP panel for TWS

Many panels in the ISPF dialogs have context-sensitive help available by pressing the F1 key. When using help, if the first panel indicates it is one of several, pressing the Enter key displays the other panels in sequence. To return to the panel from which Help was requested, press the F3 key.

The TWS JSC windows

On entering the JSC, the initial view is the window shown in Figure 16-19.

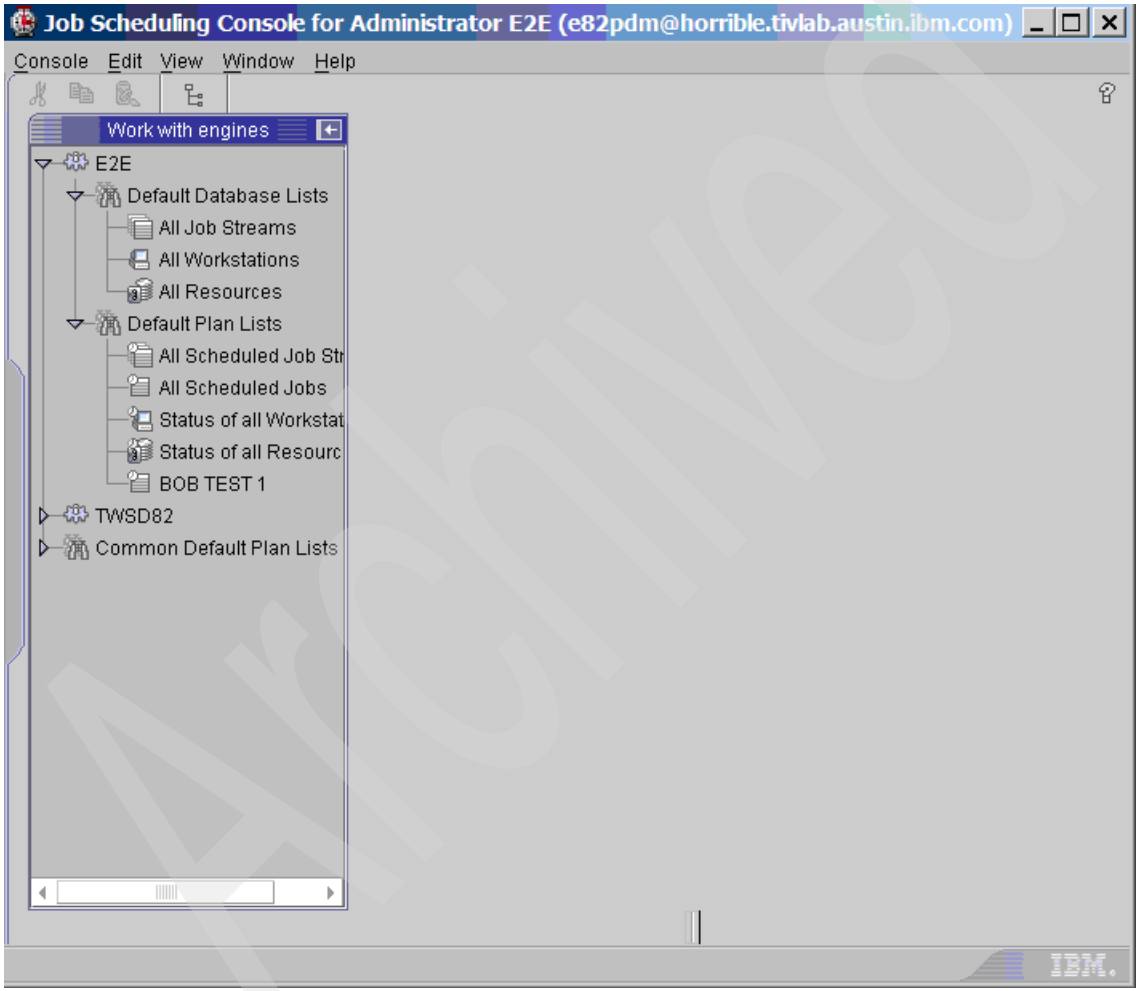


Figure 16-19 The initial JSC window

The first time the JSC is started and every time thereafter that a user opens the JSC (until the **Don't show this window again** box is checked), the user is offered the JSC tutorial (Figure 16-20). We strongly recommend that beginning JSC users use the tutorial, as it is the quickest way to learn the basics of JSC usage and navigation.

If the user needs to revisit the tutorial, it can be accessed by searching on “tutorial” under the JSC Help menu.

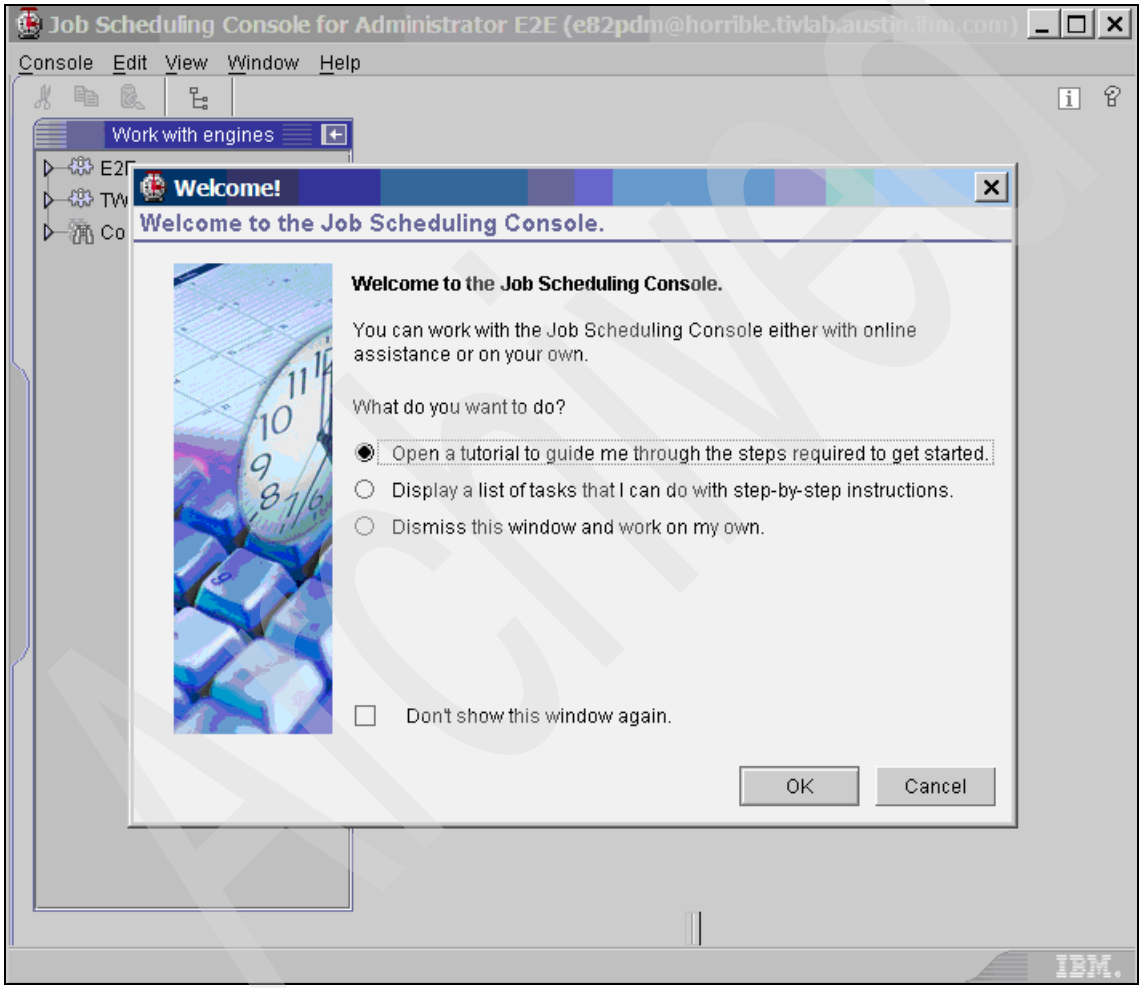


Figure 16-20 Invitation to use the JSC tutorial

The tutorial guides users through frequently utilized tasks. For many views, context-sensitive help can be accessed by the Help menu by opening the Task Assistant. The Task Assistant display automatically updates as you move from one view to another, and as you select objects within each view (Figure 16-21).

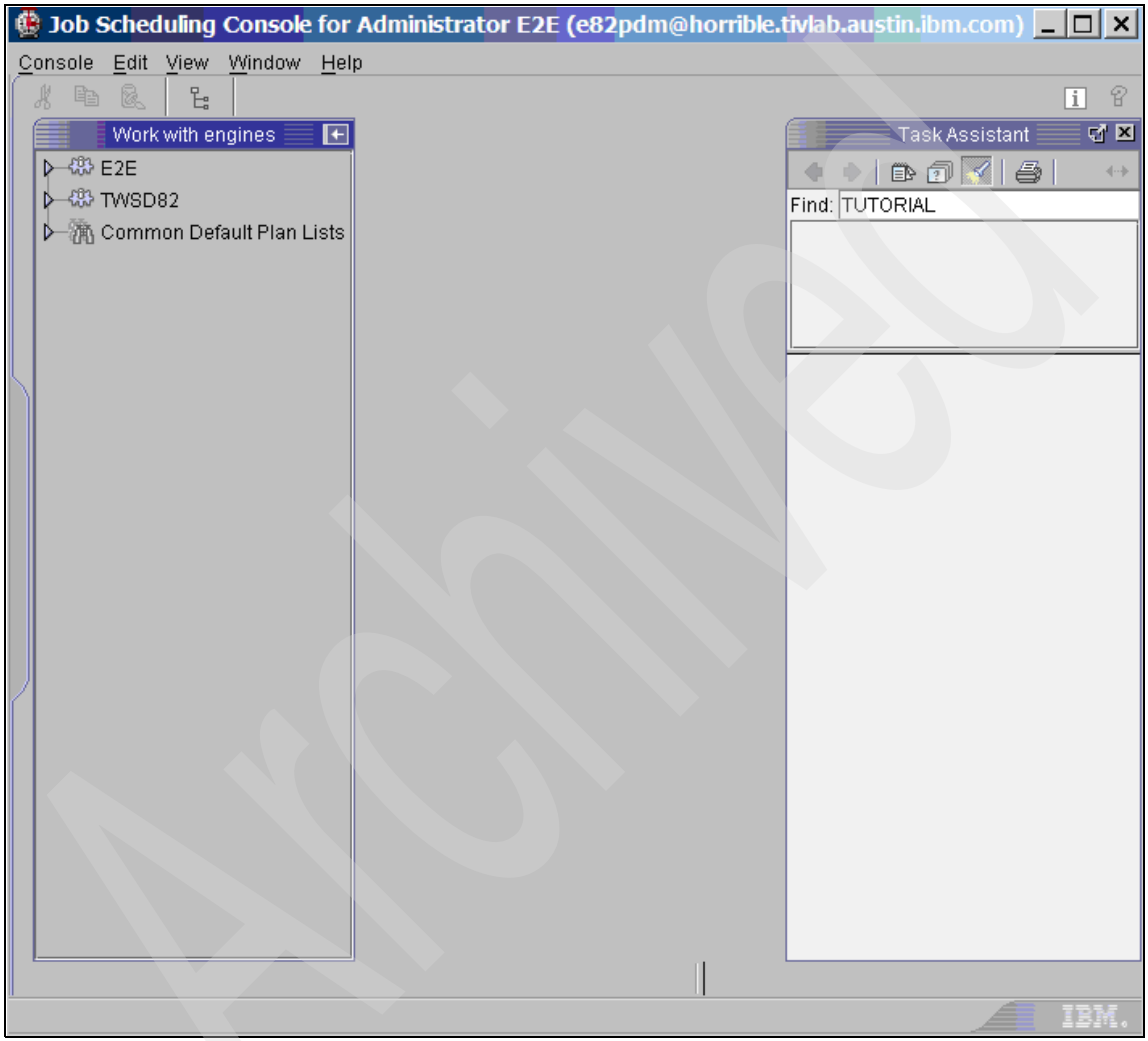


Figure 16-21 Task Assistant

If the Task Assistant obscures the user's view, it can be detached into a separate window and moved out of the way.

16.5.2 Managing applications and operations in Tivoli Workload Scheduler for z/OS end-to-end scheduling

The most common work done by schedulers is to manage applications (or jobstreams) and operations (jobs). Here we compare ISPF and JSC views, first for applications and then for operations.

Managing applications

When planning work in Tivoli Workload Scheduler for z/OS, the mainframe scheduler modifies the current plan using the ISPF dialog option 5.2 (EQQMOCLL - List Occurrences) as Figure 16-22 shows. This corresponds with the JSC All Scheduled Jobstreams view that is shown in Figure 16-23 on page 513. In Figure 16-22 and Figure 16-23 on page 513, the ISPF dialog and the JSC view have been customized to display similar information.

Figure 16-22 shows the EQQMOCLL modifying occurrences in the current plan.

```
EQQMOCLL --------- MODIFYING OCCURRENCES IN THE CURRENT PLAN Row 1 to 20 of 23

Enter the CREATE command to add a new occurrence or
enter the GRAPH command to display occurrence list graphically or
enter any of the row commands below:
B - Browse, D - Delete, M - Modify, RG - Remove from group, DG - Delete Group,
C - Complete, W -Set to Waiting, R - Rerun, CG - Complete Group

Row Application                           Input arrival   S   P  G   Add
cmd id                text                date     time             func
''' MVSXAGENT#TSTOPC  MVS XAGENT OP82 MVSOPC  06/01/31 13.00   E   9  N
''' FTA#NT#INTERACTV  TW82 NT wkstn daily test 06/03/01 13.01  W   9  N   D
''' WLMTESTC                                  06/03/01 17.00   E   9  N   D
''' AD#HOC#BR14       On Demand Dummy         06/03/09 10.00   E   4  N   D
''' MARCO#BARATTINI   pmr 40273,033,000.      06/03/19 00.00   E   5  N
''' JOBLOG                                    06/03/24 17.54   E   9  N   D
''' JOBLOG                                    06/03/24 18.05   E   9  N   D
''' JOBLOG                                    06/03/24 18.43   E   9  N   D
''' JOBLOG                                    06/03/24 18.50   E   9  N   D
''' JOBLOG                                    06/03/24 18.59   E   9  N   D
''' MVSXAGENT#TSTOPC  MVS XAGENT OP82 MVSOPC  06/03/24 13.00   E   9  N
''' MARCO#BARATTINI   pmr 40273,033,000.      06/03/26 00.00   E   5  N
''' MARCO#BARATTINI2  pmr 40273 jsc 1.4       06/03/26 00.00   E   5  N
''' MVSXAGENT#TSTOPC  MVS XAGENT OP82 MVSOPC  06/03/28 13.00   E   9  N
''' FTANTTST          TW82 NT wkstn daily test 06/03/29 14.00  E   9  N
''' FTANTTST          TW82 NT wkstn daily test 06/03/30 14.00  E   9  N
''' FTAOVER           scrptlib override       06/03/30 13.00   C   9  N
''' FTATEST                                   06/03/30 23.35   W   9  N
''' MVSXAGENT#TSTMVS  MVS XAGENT ZX82 JESMVS  06/03/30 12.01   C   4  N
''' MVSXAGENT#TSTOPC  MVS XAGENT OP82 MVSOPC  06/03/30 13.00   C   9  N   D

Command ===> _                                       Scroll ===> CSR
```

Figure 16-22 EQQMOCLL: Modifying occurrences in the current plan

In this JSC example, the columns have been moved and sort order selected so as to match the ISPF dialog panel. Not all fields in the ISPF panel have a corresponding equivalent in the JSC. Any data that is visible in the ISPF dialogs that does not show on the JSC list view can be seen by opening the object in question using a right-click on the list entry.

Figure 16-23 shows the All Scheduled Job Streams view of the JSC.

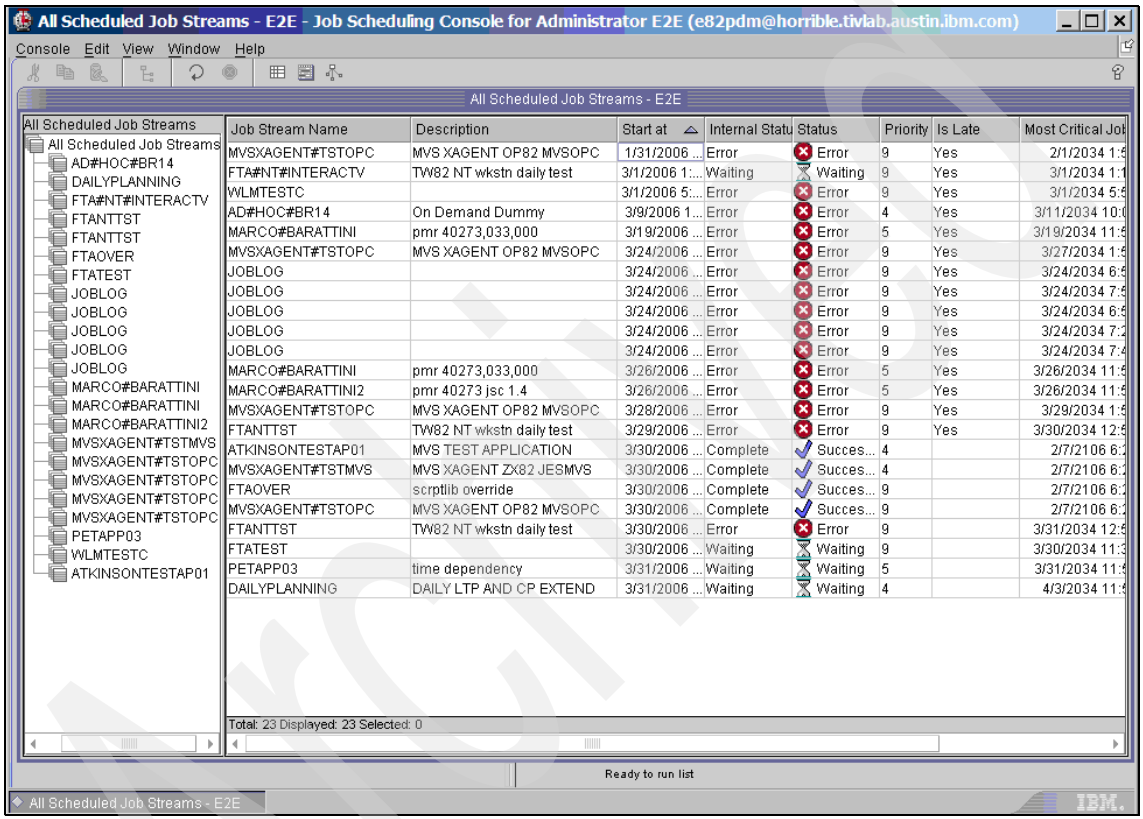


Figure 16-23 JSC All Scheduled Job Streams view

Managing operations

The ISPF dialog option 5.3 (EQQMOPRL - List Operations) shown in Figure 16-24 corresponds with the JSC All Scheduled Jobs view shown in Figure 16-25 on page 515. As in the previous comparison, the JSC view has been customized and sorted to match the ISPF panel.

Figure 16-24 shows the EQQMOPRL modifying operations in the current plan.

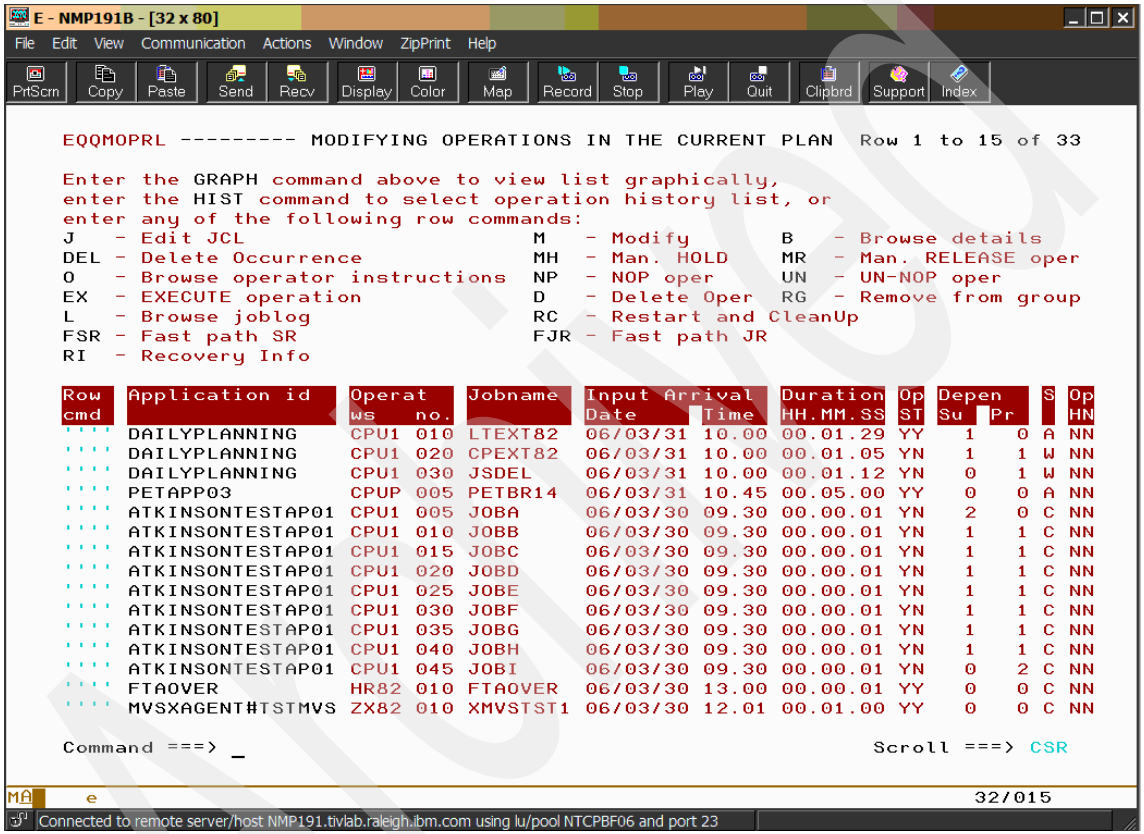


Figure 16-24 EQQMOPRL: Modifying operations in the current plan

When changing the JSC view, any column can be moved simply by dragging the column title and resized by grabbing and dragging the edge of the title. Also, most column titles will display sort arrows when hovered over with the mouse.

Figure 16-25 shows the All Scheduled Jobs view.

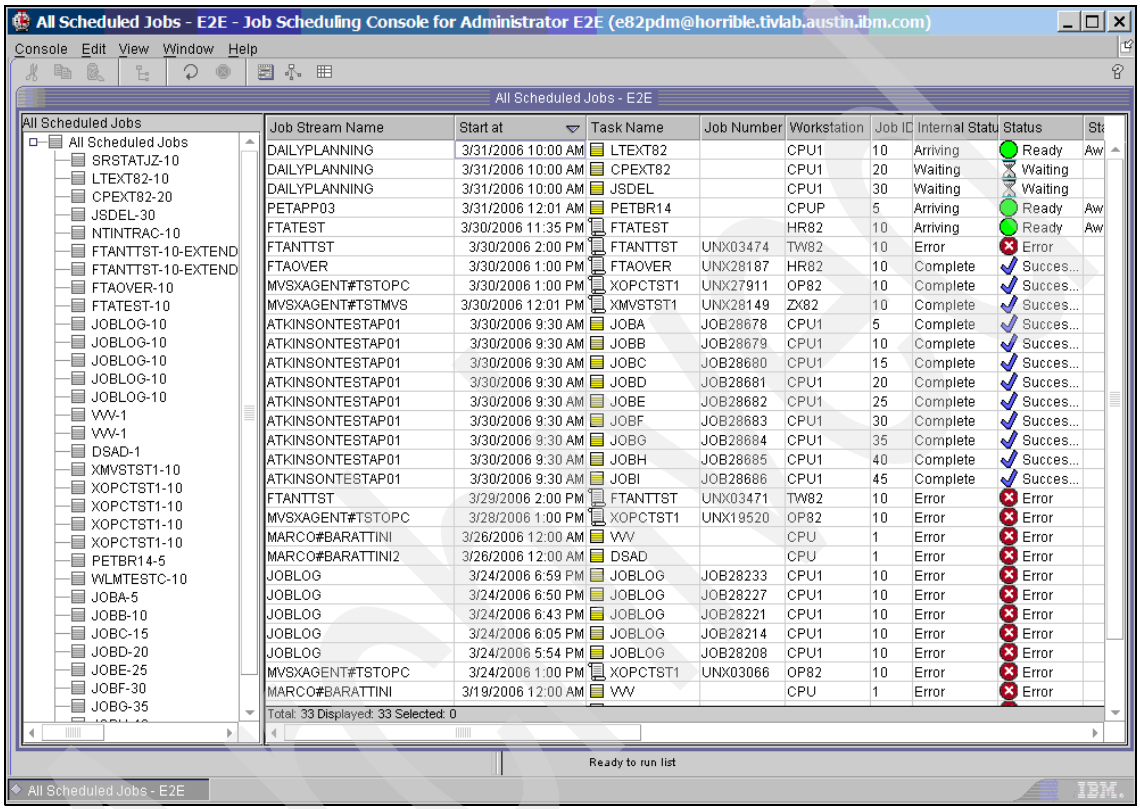


Figure 16-25 All Scheduled Jobs view

16.5.3 Comparison: building applications in ISPF and JSC

In the ISPF screens, all dependencies are defined in terms of predecessors. In this example (Figure 16-26), operation 005 is a predecessor to operation 010, and so forth.

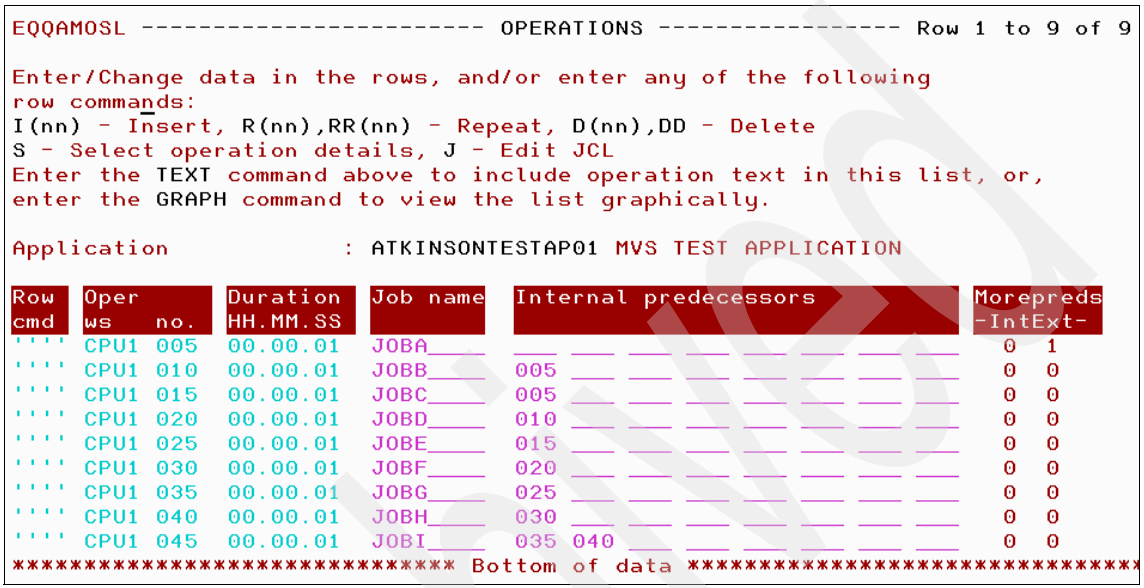


Figure 16-26 EQQAMOSL (operations or dependencies)

But in the JSC, you define predecessors with the link starting at the predecessor and pointing at the successor. The first time we defined an application with the JSC, it ran backwards. When using the JSC, remember to follow the arrows to the final job or jobs, as Figure 16-27 shows.

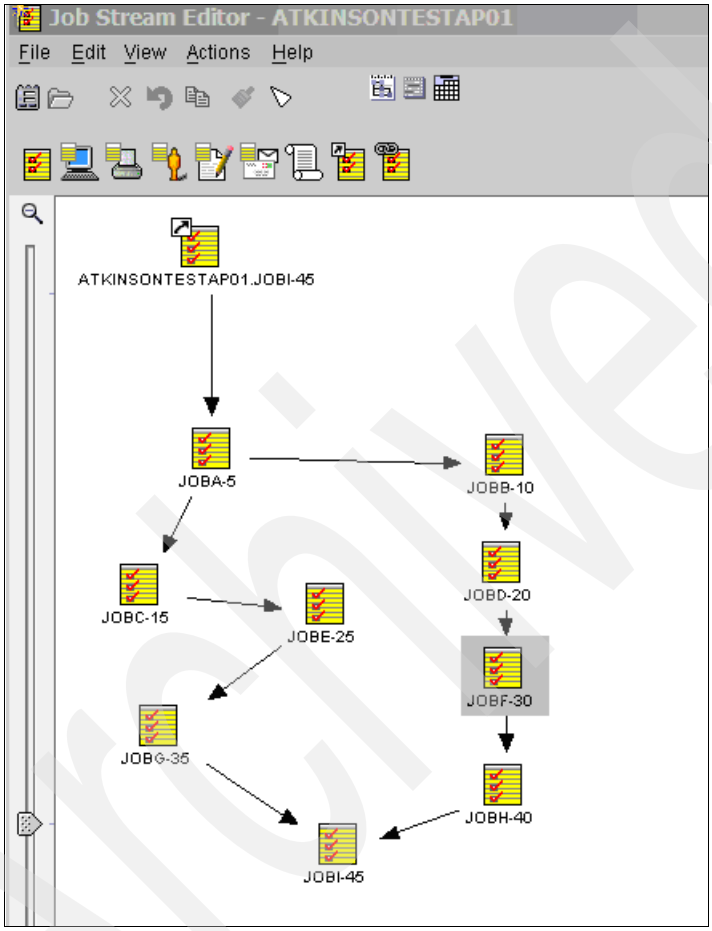


Figure 16-27 Job stream with predecessors and successors

Troubleshooting while building an application

The application in Figure 16-28 was created by a scheduler at a customer site. By looking at it using both the ISPF panels and the JSC, the relationships of the operations can be simplified.

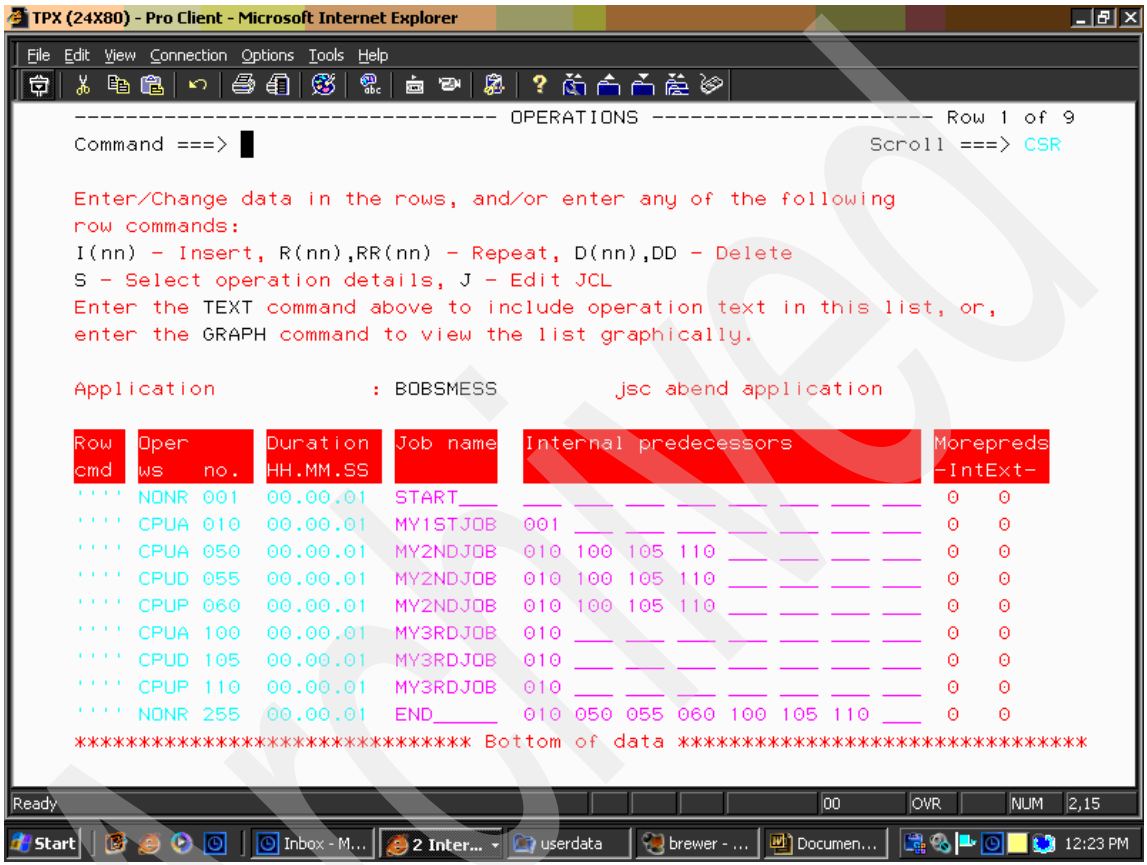


Figure 16-28 Building a complicated sample application with an ISPF panel

In the JSC, the application in Figure 16-28 shows a complicated interaction of successors and predecessors.

Figure 16-29 shows the viewing of a complicated sample application with the help of the JSC.

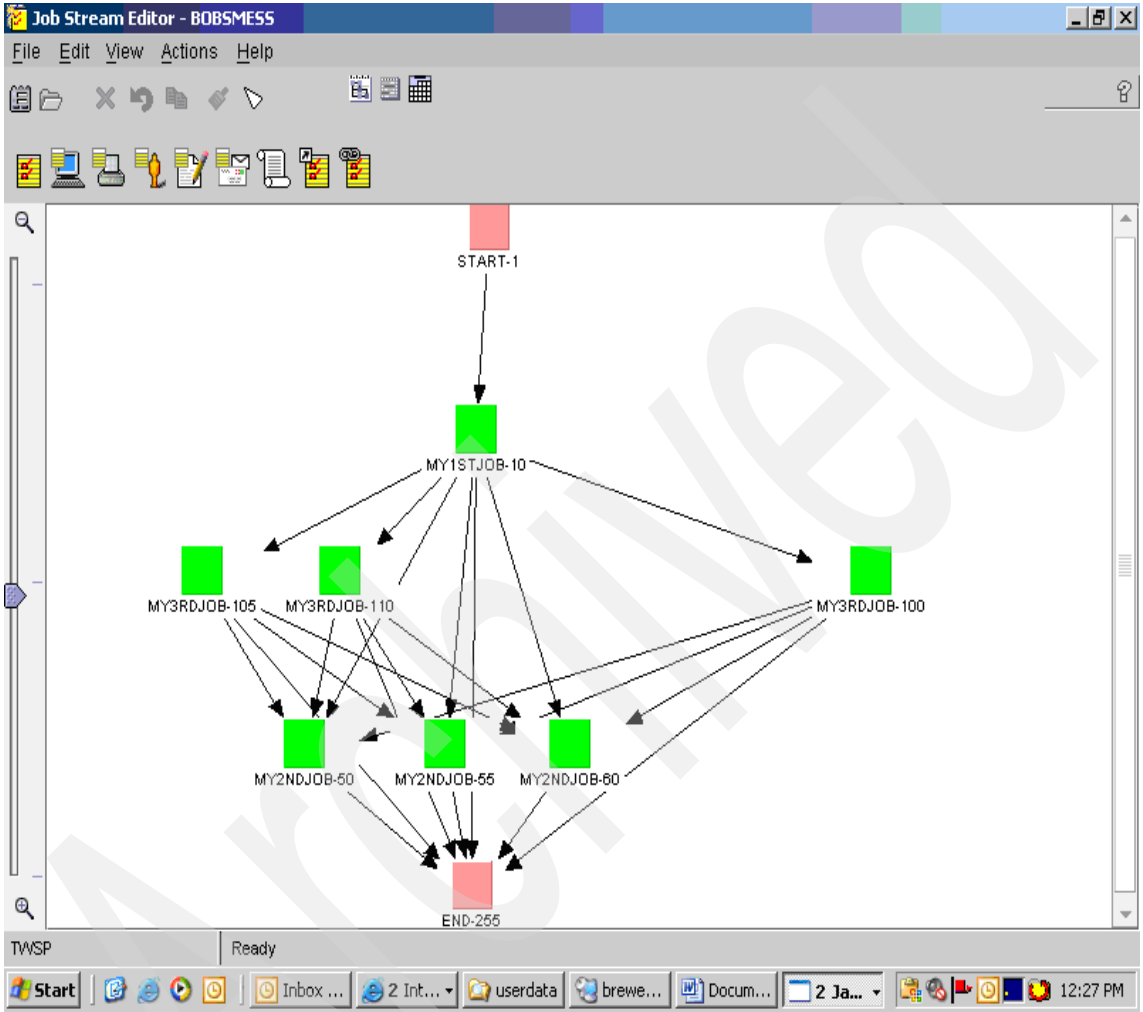


Figure 16-29 Viewing a complicated sample application with the JSC

By cleaning up the application, the JSC view is created (Figure 16-30).

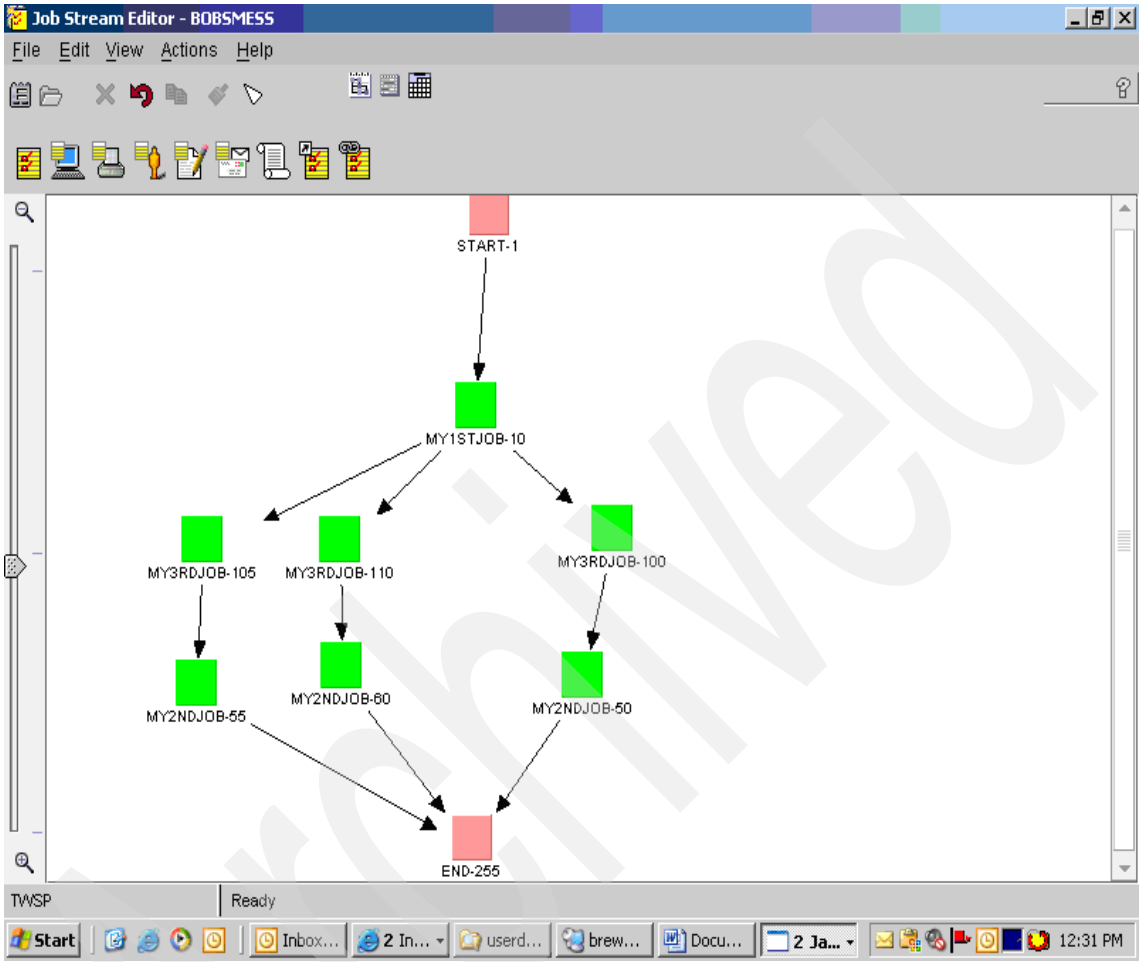


Figure 16-30 Viewing a simplified application with the JSC

The cleaned up version in ISPF, as depicted in Figure 16-31, shows a much simpler series of predecessors.

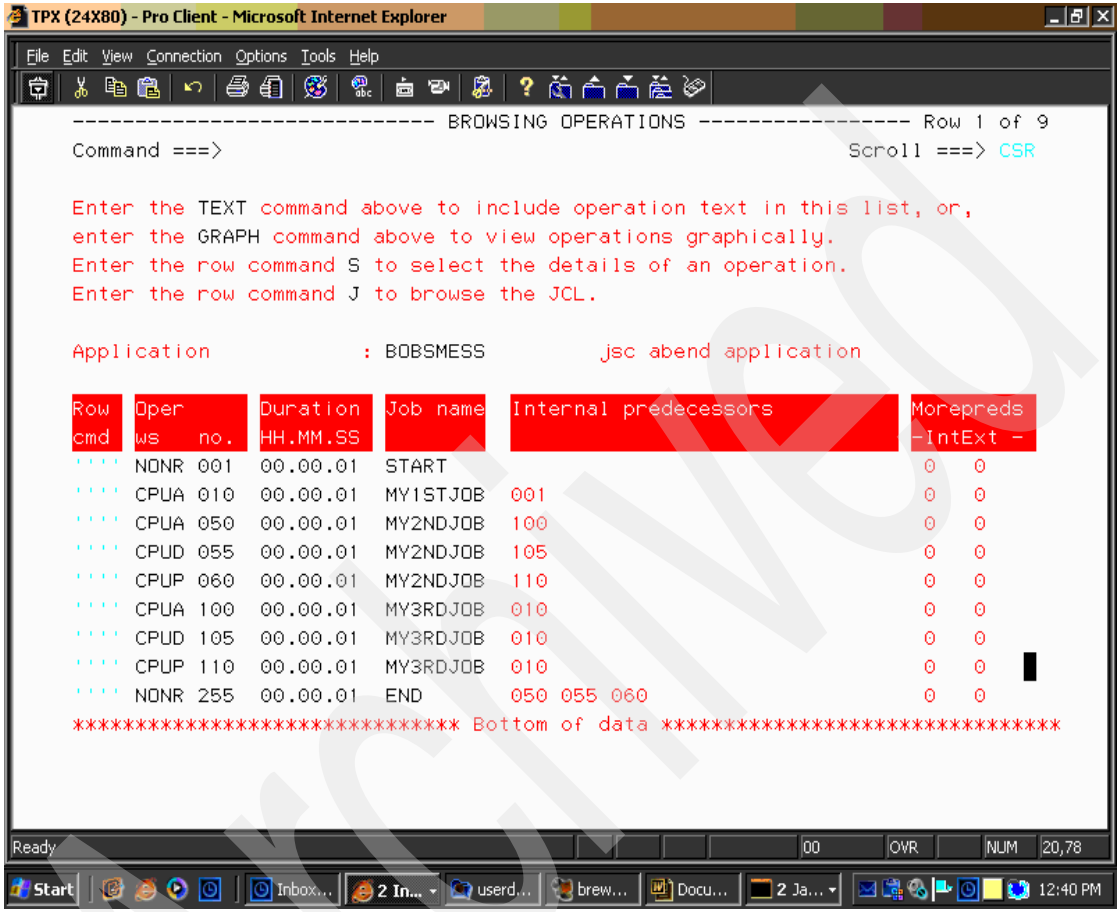


Figure 16-31 Viewing a simplified application with the ISPF panel

16.5.4 Editing JCL with the ISPF Panels and the JSC

JCL, or Job Control Language, is the scripting language for running processes on the mainframe. JCL can be edited using either the ISPF panels or with the JSC as Figure 16-32 shows.

```
EQQMJCLE ----------- EDITING JCL FOR A COMPUTER OPERATION --------------------

Edit JCL below and press END to finish or CANCEL to reject:

Application            : DAILYPLANNING     DAILY LTP AND CP EXTEND
Operation              : CPU1 30
Status of operation    : Completed
Jobname                : JSDEL             JCL last updated by: SVIOLA

****** ***************************** Top of Data ******************************
==MSG> -CAUTION- Profile changed to NUMBER OFF (from NUMBER ON STD).
==MSG>           Data does not have valid standard numbers.
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001 //JSDEL JOB MSGCLASS=X,NOTIFY=SVIOLA
000002 //*%OPC SCAN
000003 //*)OPC SETVAR TZPX=(CYYMMDD-2WK)
000004 //* NEW END DATE IS 060313
000005 //PIFCMD   EXEC PGM=EQQYCAIN,REGION=0M,PARM='082C,MSGOFF'
000006 //*STEPLIB  DD DSN=TWSZ.V8R2M0.SEQQLMD0,DISP=SHR
000007 //EQQMLIB  DD DISP=SHR,DSN=TWSZ.V8R2M0.SEQQMSG0
000008 //ERREUR   DD SYSOUT=*
000009 //EQQMLOG  DD SYSOUT=*
000010 //EQQDUMP  DD SYSOUT=*
000011 //SYSPRINT DD SYSOUT=*
000012 //SYSUDUMP DD SYSOUT=*
000013 //SYSIN    DD *
000014 ACTION=OPTIONS,BL=N,BLPRT=N,LTP=N;
000015 ACTION=DELETE,RESOURCE=JSCOM,ADID=*,IA=0603130000.
000016 //*  ACTION=DELETE,RESOURCE=JSCOM,ADID=*,IA=0603130000.
****** ***************************** Bottom of Data ***************************
Command ===> _                                              Scroll ===> CSR
```

Figure 16-32 Editing JCL with the EQQMJCLE Panel

When editing JCL or a *centralized script* via the JSC, remember that you do not have access to any of the ISPF EDIT functions to access any data set or member other than the one in EQQQJBLIB (or fetched by EQQUX002) for the scheduled operation.

Figure 16-33 shows editing JCL with the JSC Edit Jobs view.

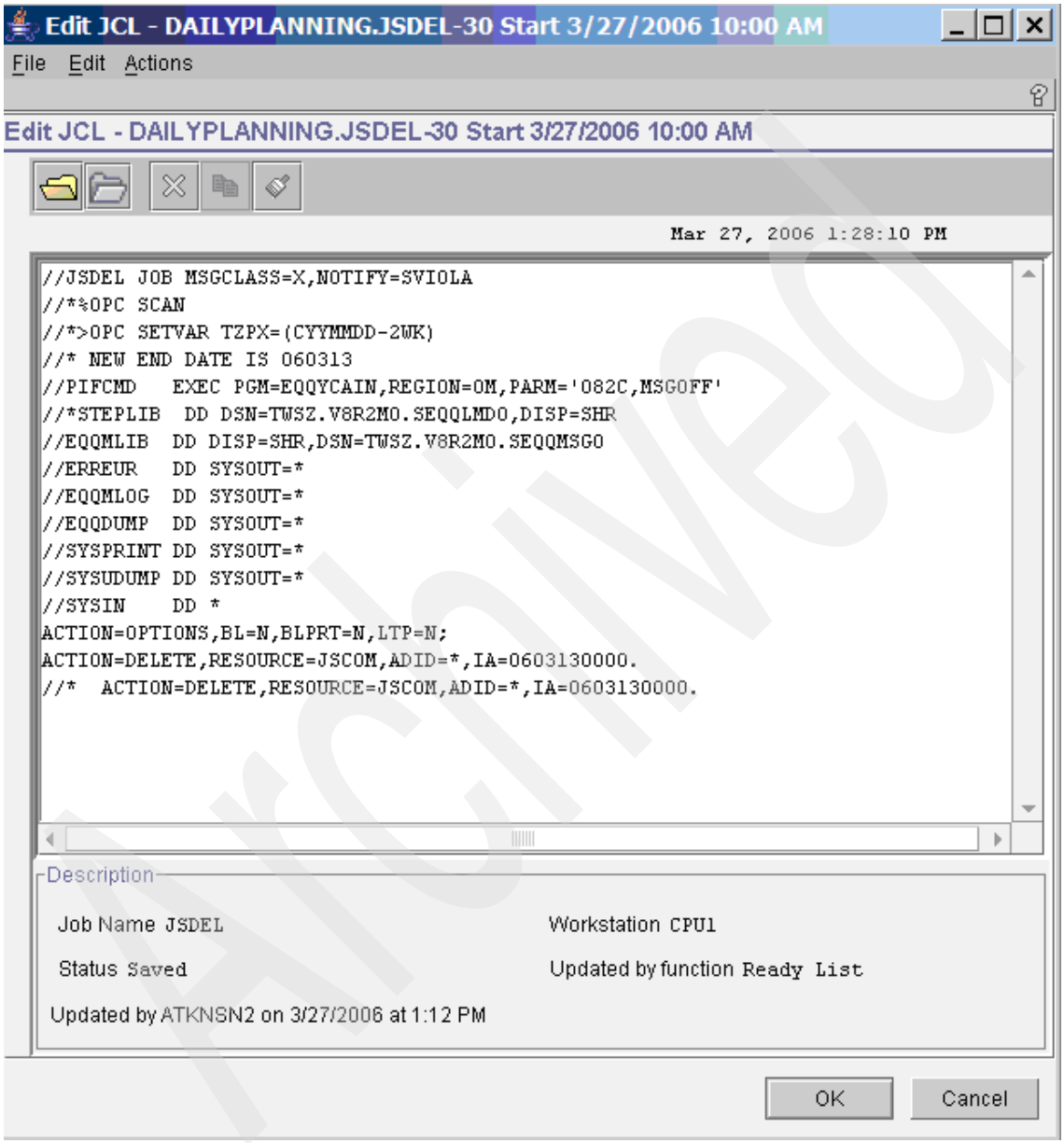


Figure 16-33 Editing JCL with the JSC Edit Jobs view

16.5.5 Viewing run cycles with the ISPF panels and JSC

Applications are planned using *run cycles*. This application has four run cycles (Figure 16-34).

```
EQQABRPL -------------------- BROWSING RUN CYCLES ------------ Row 1 to 4 of 4

Enter S row command to select the run days of a period based run cycle,
or show the rule definition for a rule based run cycle.

Application             : ATKINSONTESTAP01 MVS TEST APPLICATION
     Name of
Row  period/rule Input   Deadline       F Day In       Out of
cmd  Text        time    day time Type  rule  effect   effect   Variable table
'    MOWEDFRI    09.30 00  09.45 R     4     00/01/01 71/12/31
     Monday, Wednesday, and Friday, if work days

'    GLENDA#A    09.30 00  09.45 R     3     00/01/01 71/12/31
     Fourth day of every month - Run ON freeday

'    GLENDA#B    09.30 00  09.45 E     3     00/01/01 71/12/31
     NEVER ON FIRST SATURDAY

'    NOT#IDES    09.30 00  09.45 E     3     03/01/01 71/12/31
     NEVER RUN ON THE 15th of any month

******************************** Bottom of data ********************************
```

Figure 16-34 EQQABRPL: Four run cycle application

In the following ISPF panels, you can see exactly how each of these rules will resolve via the GENDAYS command. The first rule: MOWEDFRI schedules (Figure 16-35).

```
EQQRULSL ------------------ LIST OF GENERATED DATES --------------------------
                                                          Goto Year ===>
Rule         : MOWEDFRI    Monday, Wednesday, and Friday, if work days
Calendar     : DEFAULT              Work day end time: 00.00
Interval     : 06/01/01 - 09/12/31     Free day rule: 4

      January                2006  February               2006  March                2006
      Mo Tu We Th Fr Sa Su        Mo Tu We Th Fr Sa Su        Mo Tu We Th Fr Sa Su
                        01              01 02 03 04 05              01 02 03 04 05
      02 03 04 05 06 07 08        06 07 08 09 10 11 12        06 07 08 09 10 11 12
      09 10 11 12 13 14 15        13 14 15 16 17 18 19        13 14 15 16 17 18 19
      16 17 18 19 20 21 22        20 21 22 23 24 25 26        20 21 22 23 24 25 26
      23 24 25 26 27 28 29        27 28                       27 28 29 30 31
      30 31
      April                  2006  May                  2006  June                 2006
      Mo Tu We Th Fr Sa Su        Mo Tu We Th Fr Sa Su        Mo Tu We Th Fr Sa Su
                     01 02        01 02 03 04 05 06 07              01 02 03 04
      03 04 05 06 07 08 09        08 09 10 11 12 13 14        05 06 07 08 09 10 11
      10 11 12 13 14 15 16        15 16 17 18 19 20 21        12 13 14 15 16 17 18
      17 18 19 20 21 22 23        22 23 24 25 26 27 28        19 20 21 22 23 24 25
      24 25 26 27 28 29 30        29 30 31                    26 27 28 29 30

                              VALID RULE

Command ===>                                                Scroll ===> CSR
```

Figure 16-35 EQQRULSL: List of generated dates for rule MOWEDFRI

And the second rule, GLENDA#A schedules (Figure 16-36).

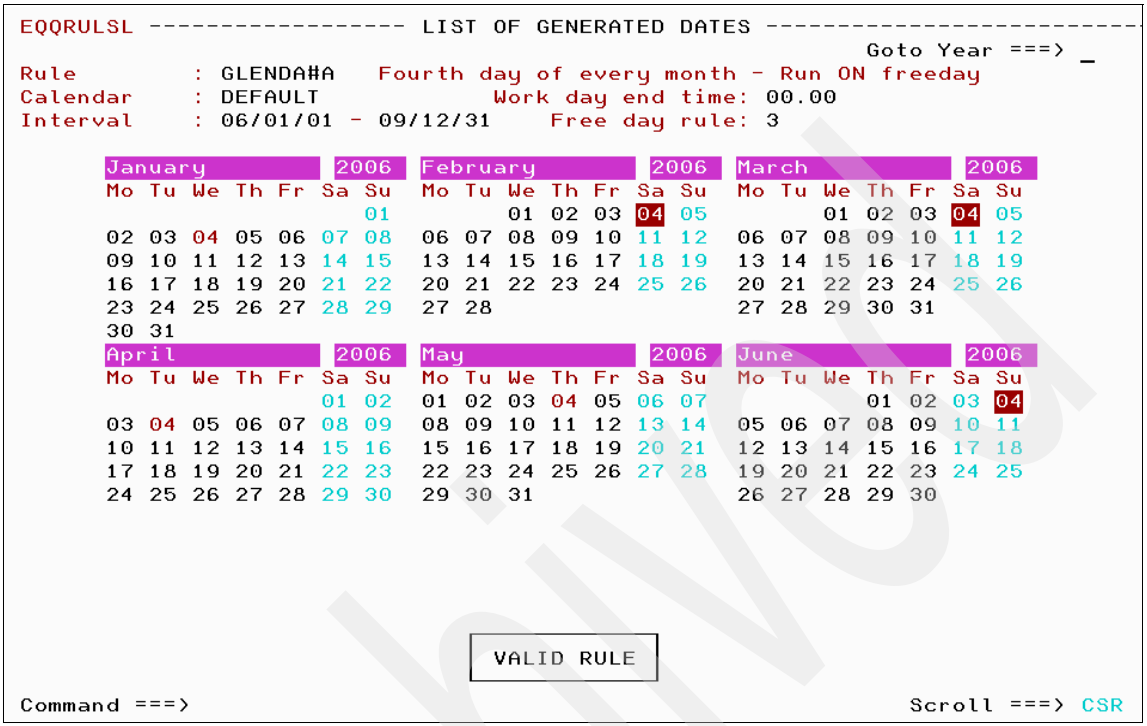


Figure 16-36 EQQRULSL: List of generated dates for rule GLENDA#A

So, with the ISPF panels, a user can view only one run cycle rule at a time. If the environment has a combination of normal and exclusion run cycles on every Monday, Wednesday, and Friday but not on the fourth of the month, it is very difficult to visualize how they will *net out*. There is a batch job called *Calculate and print run cycles* (EQQADCOP) but it only works within the time covered by the existing Long Term Plan.

By comparison, if you select the Calendar view in the JSC, the total effect of all the run cycles for the application is shown in Figure 16-37, and you can see which of those dates is generated by a given run cycle by selecting that one in the list.

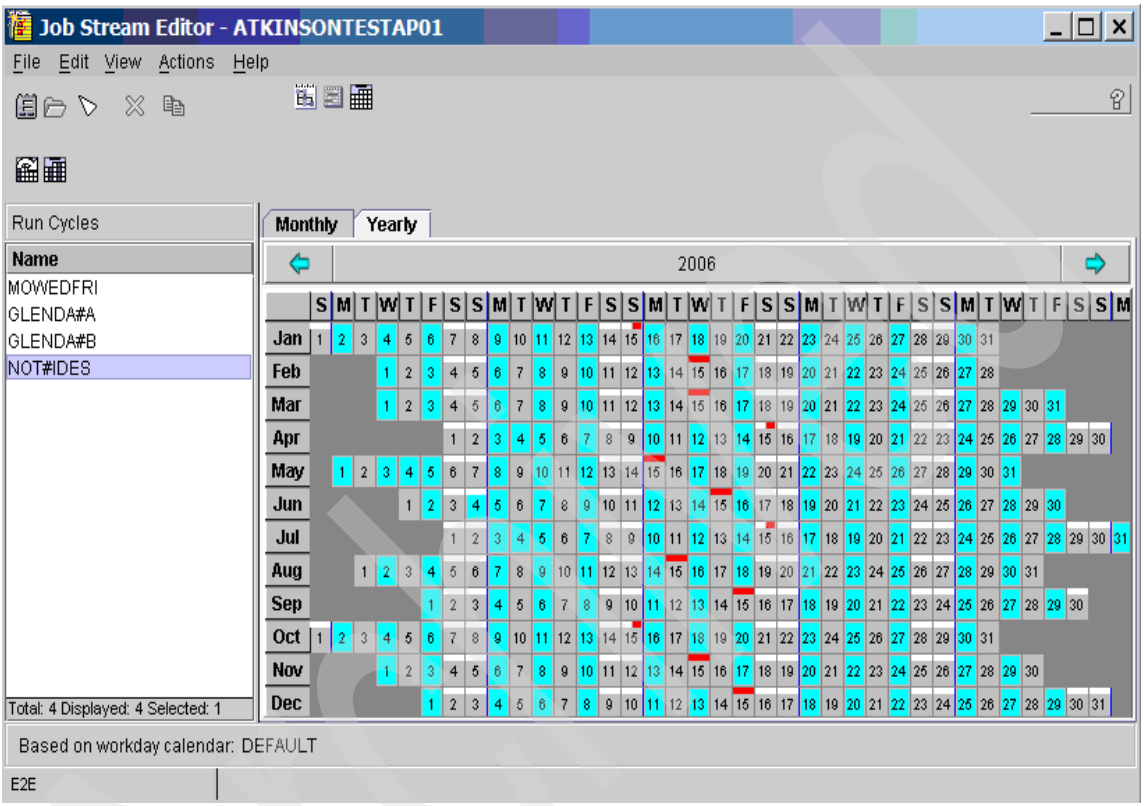


Figure 16-37 Calendar View in the Job Stream Editor

17

End-to-end scheduling scenarios

In this chapter, we describe different scenarios and examples for Tivoli Workload Scheduler for z/OS end-to-end scheduling.

We describe and show:

- Description of our environment and systems
- Creation of the Symphony file in detail
- Migrating Tivoli OPC tracker agents to end-to-end scheduling
- Conversion from Tivoli Workload Scheduler network to Tivoli Workload Scheduler for z/OS managed network
- Tivoli Workload Scheduler for z/OS end-to-end fail-over scenarios
- Backup and maintenance guidelines for FTAs
- Security on fault-tolerant agents
- End-to-end scheduling tips and tricks

17.1 Description of our environment and systems

In this section, we describe the systems and configuration we used for the end-to-end test scenarios when working on this book.

Figure 17-1 shows the systems and configuration that are used for the end-to-end scenarios. All of the systems are connected using TCP/IP connections.

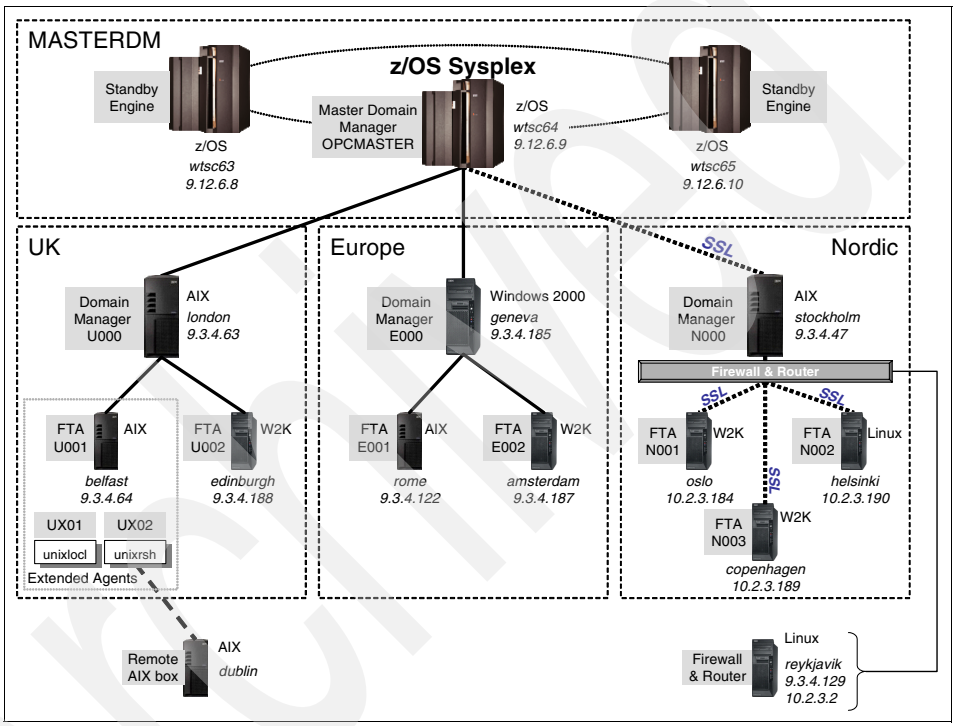


Figure 17-1 Systems and configuration used in end-to-end scheduling test scenarios

We defined the following started task procedure names in z/OS:

TWST	For the Tivoli Workload Scheduler for z/OS agent
TWSC	For the Tivoli Workload Scheduler for z/OS engine
TWSCE2E	For the end-to-end server
TWSCJSC	For the Job Scheduling Console server

In the following sections we have listed the started task procedure for our end-to-end server and the different initialization statements defined for the end-to-end scheduling network in Figure 17-1.

Started task procedure for the end-to-end server (TWSCE2E)

Example 17-1 shows the started task procedure for the Tivoli Workload Scheduler for z/OS end-to-end server, TWSCE2E.

Example 17-1 Started task procedure for the end-to-end server TWSCE2E

```
//TWSCE2E   EXEC PGM=EQQSERVR,REGION=64M,TIME=1440
//*  NOTE:  64M IS THE MINIMUM REGION SIZE FOR E2E (SEE PQ78043)
//*********************************************************************
//* THIS IS A STARTED TASK PROCEDURE FOR AN OPC SERVER DEDICATED
//* FOR END-TO-END SCHEDULING.
//*********************************************************************
//STEPLIB   DD  DISP=SHR,DSN=EQQ.SEQQLMD0
//EQQMLIB   DD  DISP=SHR,DSN=EQQ.SEQQMSG0
//EQQMLOG   DD  SYSOUT=*
//EQQPARM   DD  DISP=SHR,DSN=TWS.INST.PARM(TWSCE2E)
//SYSMDUMP  DD  DISP=SHR,DSN=TWS.INST.SYSDUMPS
//EQQDUMP   DD  DISP=SHR,DSN=TWS.INST.EQQDUMPS
//EQQTWSIN  DD  DISP=SHR,DSN=TWS.INST.TWSC.TWSIN -> INPUT TO CONTROLLER
//EQQTWSOU  DD  DISP=SHR,DSN=TWS.INST.TWSC.TWSOU -> OUTPUT FROM CONT.
//EQQTWSCS  DD  DISP=SHR,DSN=TWS.INST.CS         -> CENTRALIZED SCRIPTS
```

The end-to-end server (TWSCE2E) initialization statements

Example 17-2 defines the initialization statements for the end-to-end scheduling network shown in Figure 17-1 on page 530.

Example 17-2 End-to-end server (TWSCE2E) initialization statements

```
/*********************************************************************/
/* SERVOPTS: run-time options for end-to-end server                   */
/*********************************************************************/
SERVOPTS   SUBSYS(TWSC)
/*-------------------------------------------------------------------*/
/* TCP/IP server is needed for end-to-end usage.                     */
/*-------------------------------------------------------------------*/
       PROTOCOL(E2E)               /* This server is for E2E "only"*/
       TPLGYPRM(TOPOLOGY)          /* E2E topology definition mbr. */
/*-------------------------------------------------------------------*/
/* If you want to use Automatic Restart manager you must specify:    */
/*-------------------------------------------------------------------*/
       ARM(YES)                    /* Use ARM to restart if abend  */
```

Example 17-3 shows the TOPOLOGY initialization statements.

Example 17-3 TOPOLOGY initialization statements; member name is TOPOLOGY

```
/*********************************************************************/
/* TOPOLOGY: End-to-End options                                      */
/*********************************************************************/
TOPOLOGY TPLGYMEM(TPDOMAIN)             /* Mbr. with domain+FTA descr.*/
         USRMEM(TPUSER)                 /* Mbr. with Windows user+pw  */
         BINDIR('/usr/lpp/TWS/V8R2M0')  /* The TWS for z/OS inst. dir */
         WRKDIR('/tws/twsce2ew')        /* The TWS for z/OS work dir  */
         LOGLINES(200)                  /* Lines sent by joblog retr. */
         TRCDAYS(10)                    /* Days to keep stdlist files */
         CODEPAGE(IBM-037)              /* Codepage for translator    */
         TCPIPJOBNAME(TCPIP)            /* Name of TCPIP started task */
         ENABLELISTSECCHK(N)            /* CHECK SEC FILE FOR LIST?   */
         PLANAUDITLEVEL(0)              /* Audit level on DMs&FTAs    */
         GRANTLOGONASBATCH(Y)           /* Automatically grant right? */
         HOSTNAME(twsce2e.itso.ibm.com) /* DNS hostname for server    */
         PORTNUMBER(31111)              /* Port for netman in USS     */
```

Example 17-4 shows DOMREC and CPUREC initialization statements for the network in Figure 17-1 on page 530.

Example 17-4 Domain and fault-tolerant agent definitions; member name is TPDOMAIN

```
/*********************************************************************/
/* DOMREC: Defines the domains in the distributed Tivoli Workload    */
/*         Scheduler network                                         */
/*********************************************************************/
/*-------------------------------------------------------------------*/
/* Specify one DOMREC for each domain in the distributed network.    */
/* With the exception of the master domain (whose name is MASTERDM   */
/* and consist of the TWS for z/OS controller).                      */
/*-------------------------------------------------------------------*/
DOMREC   DOMAIN(UK)                 /* Domain name   = UK            */
         DOMMNGR(U000)              /* Domain manager= FLORENCE      */
         DOMPARENT(MASTERDM)        /* Domain parent = MASTERDM      */
DOMREC   DOMAIN(Europe)             /* Domain name   = Europe        */
         DOMMNGR(E000)              /* Domain manager= Geneva        */
         DOMPARENT(MASTERDM)        /* Domain parent = MASTERDM      */
DOMREC   DOMAIN(Nordic)             /* Domain name   = Nordic        */
         DOMMNGR(N000)              /* Domain manager= Stockholm     */
         DOMPARENT(MASTERDM)        /* Domain parent = MASTERDM      */
/*********************************************************************/
/*********************************************************************/
```

```
/* CPUREC: Defines the workstations in the distributed Tivoli         */
/*         Workload Scheduler network                                 */
/*********************************************************************/
/*-------------------------------------------------------------------*/
/* You must specify one CPUREC for workstation in the TWS network    */
/* with the exception of OPC Controller which acts as Master Domain  */
/* Manager                                                           */
/*-------------------------------------------------------------------*/
CPUREC   CPUNAME(U000)               /* DM of UK domain              */
         CPUOS(AIX)/* AIX operating system                    */
         CPUNODE(london.itsc.austin.ibm.com) /* Hostname of CPU     */
         CPUTCPIP(31182)             /* TCP port number of NETMAN    */
         CPUDOMAIN(UK)               /* The TWS domain name for CPU  */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT  */
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU  */
         CPUFULLSTAT(ON)             /* Full status on for DM        */
         CPURESDEP(ON)               /* Resolve dependencies on for DM*/
         CPULIMIT(20)                /* Number of jobs in parallel   */
         CPUTZ(CST)                  /* Time zone for this CPU       */
         CPUUSER(maestro)            /* Default user for jobs on CPU */
CPUREC   CPUNAME(E000)               /* DM of Europe domain          */
         CPUOS(WNT)                  /* Windows 2000 operating system */
         CPUNODE(geneva.itsc.austin.ibm.com)      /* Hostname of CPU */
         CPUTCPIP(31182)             /* TCP port number of NETMAN    */
         CPUDOMAIN(Europe)           /* The TWS domain name for CPU  */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT  */
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU  */
         CPUFULLSTAT(ON)             /* Full status on for DM        */
         CPURESDEP(ON)               /* Resolve dependencies on for DM*/
         CPULIMIT(20)                /* Number of jobs in parallel   */
         CPUTZ(CST)                  /* Time zone for this CPU       */
         CPUUSER(tws)                /* Default user for jobs on CPU */
CPUREC   CPUNAME(N000)               /* DM of Nordic domain          */
         CPUOS(AIX)                  /* AIX operating system         */
         CPUNODE(stockholm.itsc.austin.ibm.com)    /* Hostname of CPU */
         CPUTCPIP(31182)             /* TCP port number of NETMAN    */
         CPUDOMAIN(Nordic)           /* The TWS domain name for CPU  */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT  */
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU  */
         CPUFULLSTAT(ON)             /* Full status on for DM        */
         CPURESDEP(ON)               /* Resolve dependencies on for DM*/
         CPULIMIT(20)                /* Number of jobs in parallel   */
         CPUTZ(CST)                  /* Time zone for this CPU       */
         CPUUSER(tws)                /* Default user for jobs on CPU */
CPUREC   CPUNAME(U001)               /* 1st FTA in UK domain         */
```

```
         CPUOS(AIX)                  /* AIX operating system          */
         CPUNODE(belfast.itsc.austin.ibm.com)      /* Hostname of CPU */
         CPUTCPIP(31182)             /* TCP port number of NETMAN     */
         CPUDOMAIN(UK)               /* The TWS domain name for CPU   */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT   */
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU   */
         CPUFULLSTAT(OFF)            /* Full status off for FTA       */
         CPURESDEP(OFF)              /* Resolve dep. off for FTA      */
         CPULIMIT(20)                /* Number of jobs in parallel    */
         CPUSERVER(1)                /* Not allowed for DM/XAGENT CPU */
         CPUTZ(CST)                  /* Time zone for this CPU        */
         CPUUSER(tws)                /* Default user for jobs on CPU  */
CPUREC   CPUNAME(UOO2)               /* 2nd FTA in UK domain          */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT   */
         CPUOS(WNT)                  /* Windows 2000 operating system */
         CPUNODE(edinburgh.itsc.austin.ibm.com)    /* Hostname of CPU */
         CPUTCPIP(31182)             /* TCP port number of NETMAN     */
         CPUDOMAIN(UK)               /* The TWS domain name for CPU   */
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU   */
         CPUFULLSTAT(OFF)            /* Full status off for FTA       */
         CPURESDEP(OFF)              /* Resolve dep. off for FTA      */
         CPULIMIT(20)                /* Number of jobs in parallel    */
         CPUSERVER(2)                /* Not allowed for DM/XAGENT CPU */
         CPUTZ(CST)                  /* Time zone for this CPU        */
         CPUUSER(tws)                /* Default user for jobs on CPU  */
         CPUUSER(tws)                /* Default user for jobs on CPU  */
CPUREC   CPUNAME(EOO1)               /* 1st FTA in Europe domain      */
         CPUOS(AIX)                  /* AIX operating system          */
         CPUNODE(rome.itsc.austin.ibm.com)         /* Hostname of CPU */
         CPUTCPIP(31182)             /* TCP port number of NETMAN     */
         CPUDOMAIN(Europe)           /* The TWS domain name for CPU   */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT   */
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU   */
         CPUFULLSTAT(OFF)            /* Full status off for FTA       */
         CPURESDEP(OFF)              /* Resolve dep. off for FTA      */
         CPULIMIT(20)                /* Number of jobs in parallel    */
         CPUSERVER(1)                /* Not allowed for domain mng.   */
         CPUTZ(CST)                  /* Time zone for this CPU        */
         CPUUSER(tws)                /* Default user for jobs on CPU  */
CPUREC   CPUNAME(EOO2)               /* 2nd FTA in Europe domain      */
         CPUOS(WNT)                  /* Windows 2000 operating system */
         CPUNODE(amsterdam.itsc.austin.ibm.com)    /* Hostname of CPU */
         CPUTCPIP(31182)             /* TCP port number of NETMAN     */
         CPUDOMAIN(Europe)           /* The TWS domain name for CPU   */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT   */
```

```
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU   */
         CPUFULLSTAT(OFF)            /* Full status off for FTA       */
         CPURESDEP(OFF)              /* Resolve dep. off for FTA      */
         CPULIMIT(20)                /* Number of jobs in parallel    */
         CPUSERVER(2)                /* Not allowed for domain mng.   */
        CPUTZ(CST)                  /* Time zone for this CPU        */
        CPUUSER(tws)                /* Default user for jobs on CPU  */
CPUREC   CPUNAME(NO01)               /* 1st FTA in Nordic domain      */
        CPUOS(WNT)                  /* Windows 2000 operating system */
         CPUNODE(oslo.itsc.austin.ibm.com)         /* Hostname of CPU */
         CPUTCPIP(31182)             /* TCP port number of NETMAN     */
         CPUDOMAIN(Nordic)           /* The TWS domain name for CPU   */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT   */
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU   */
         CPUFULLSTAT(OFF)            /* Full status off for FTA       */
         CPURESDEP(OFF)              /* Resolve dep. off for FTA      */
         CPULIMIT(20)                /* Number of jobs in parallel    */
         CPUSERVER(1)                /* Not allowed for domain mng.   */
         CPUTZ(CST)                  /* Time zone for this CPU        */
         CPUUSER(tws)                /* Default user for jobs on CPU  */
         SSLLEVEL(OFF)               /* Use SSL? ON/OFF/ENABLED/FORCE */
         SSLPORT(31382)              /* Port for SSL communication    */
         FIREWALL(Y)                 /* Is CPU behind a firewall?     */
CPUREC   CPUNAME(NO02)               /* 2nd FTA in Nordic domain      */
         CPUOS(UNIX)                 /* Linux operating system        */
         CPUNODE(helsinki.itsc.austin.ibm.com)     /* Hostname of CPU */
         CPUTCPIP(31182)             /* TCP port number of NETMAN     */
         CPUDOMAIN(Nordic)           /* The TWS domain name for CPU   */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT   */
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU   */
         CPUFULLSTAT(OFF)            /* Full status off for FTA       */
         CPURESDEP(OFF)              /* Resolve dep. off for FTA      */
         CPULIMIT(20)                /* Number of jobs in parallel    */
         CPUSERVER(2)                /* Not allowed for domain mng.   */
         CPUTZ(CST)                  /* Time zone for this CPU        */
         CPUUSER(tws)                /* Default user for jobs on CPU  */
         SSLLEVEL(OFF)               /* Use SSL? ON/OFF/ENABLED/FORCE */
         SSLPORT(31382)              /* Port for SSL communication    */
         FIREWALL(Y)                 /* Is CPU behind a firewall?     */
CPUREC   CPUNAME(NO03)               /* 3rd FTA in Nordic domain      */
         CPUOS(WNT)                  /* Windows 2000 operating system */
         CPUNODE(copenhagen.itsc.austin.ibm.com)   /* Hostname of CPU */
         CPUTCPIP(31182)             /* TCP port number of NETMAN     */
         CPUDOMAIN(Nordic)           /* The TWS domain name for CPU   */
         CPUTYPE(FTA)                /* CPU type: FTA/SAGENT/XAGENT   */
```

```
         CPUAUTOLNK(ON)              /* Autolink is on for this CPU   */
         CPUFULLSTAT(OFF)            /* Full status off for FTA       */
         CPURESDEP(OFF)              /* Resolve dep. off for FTA      */
         CPULIMIT(20)                /* Number of jobs in parallel    */
         CPUSERVER(3)                /* Not allowed for domain mng.   */
         CPUTZ(CST)                  /* Time zone for this CPU        */
         CPUUSER(tws)                /* Default user for jobs on CPU  */
         SSLLEVEL(OFF)               /* Use SSL? ON/OFF/ENABLED/FORCE */
         SSLPORT(31382)              /* Port for SSL communication    */
         FIREWALL(Y)                /* Is CPU behind a firewall?     */
CPUREC   CPUNAME(UX01)               /* X-agent in UK Domain          */
         CPUOS(OTHER)                /* Extended agent                */
         CPUNODE(belfast.itsc.austin.ibm.com   /* Hostname of CPU     */
         CPUDOMAIN(UK)               /* The TWS domain name for CPU   */
         CPUHOST(U001)               /* U001 is the host for x-agent  */
         CPUTYPE(XAGENT)             /* This is an extended agent     */
         CPUACCESS(unixlocl)         /* use unixlocl access method    */
         CPULIMIT(2)                 /* Number of jobs in parallel    */
         CPUTZ(CST)                  /* Time zone for this CPU        */
         CPUUSER(tws)                /* Default user for jobs on CPU  */
CPUREC   CPUNAME(UX02)               /* X-agent in UK Domain          */
         CPUOS(OTHER)                /* Extended agent                */
         CPUNODE(belfast.itsc.austin.ibm.com   /* Hostname of CPU     */
         CPUDOMAIN(UK)               /* The TWS domain name for CPU   */
         CPUHOST(U001)               /* U001 is the host for x-agent  */
         CPUTYPE(XAGENT)             /* This is an extended agent     */
         CPUACCESS(unixrsh)          /* use unixrsh access method     */
         CPULIMIT(2)                 /* Number of jobs in parallel    */
         CPUTZ(CST)                  /* Time zone for this CPU        */
         CPUUSER(tws)                /* Default user for jobs on CPU  */
```

User and password definitions for the Windows fault-tolerant workstations are defined as shown in Example 17-5.

Example 17-5 User and password definition for Windows FTAs; member name is TPUSER

```
/*********************************************************************/
/* USRREC: Windows users password definitions                        */
/*********************************************************************/
/*-------------------------------------------------------------------*/
/* You must specify at least one USRREC for each Windows workstation */
/* in the distributed TWS network.                                   */
/*-------------------------------------------------------------------*/
USRREC USRCPU(U002)
```

```
       USRNAM(tws)
       USRPSW('tws')
USRREC USRCPU(E000)
       USRNAM(tws)
       USRPSW('tws')
USRREC USRCPU(E000)
       USRNAM(tws)
       USRPSW('tws')
USRREC USRCPU(N001)
       USRNAM(tws)
       USRPSW('tws')
USRREC USRCPU(N003)
       USRNAM(tws)
       USRPSW('tws')
```

17.2 Creation of the Symphony file in detail

A new Symphony file is generated whenever any of these daily planning batch jobs is run:

- Extend the current plan
- Replan the current plan
- Renew the Symphony

Daily planning batch jobs must be able to read from and write to the HFS working directory (WRKDIR) because these jobs create the Symnew file in WRKDIR. For this reason, the group associated with WRKDIR must contain all of the users that will run daily planning batch jobs.

The end-to-end server task starts the translator process in USS (via the starter process). The translator process inherits its ownership from the starting task, so it runs as the same user as the end-to-end server task.

The translator process must be able to read from and write to the HFS working directory (WRKDIR). For this reason, WRKDIR must be owned by the user associated with the end-to-end server started task (E2ESERV in the following example). This underscores the importance of specifying the correct user and group in EQQPCS05.

Figure 17-2 on page 538 shows the steps of Symphony file creation:

1. The daily planning batch job copies the Symphony Current Plan VSAM data set to an HFS file in WRKDIR called SymUSER, where USER is the user name of the user who submitted the batch job.

2. The daily planning batch job renames SymUSER to Symnew.
3. The translator program running in UNIX System Services copies Symnew to Symphony and Sinfonia.

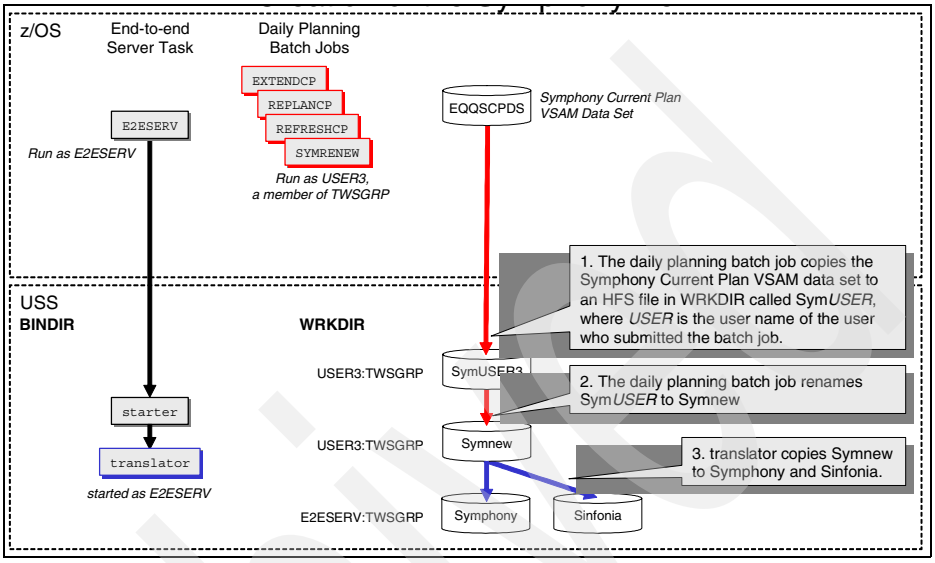


Figure 17-2 Creation of the Symphony file in WRKDIR

This illustrates how process ownership of the translator program is inherited from the end-to-end server task. The figure also shows how file ownership of Symnew and Symphony are inherited from the daily planning batch jobs and translator, respectively.

17.3 Migrating Tivoli OPC tracker agents to end-to-end scheduling

In this section, we describe how to migrate from a Tivoli OPC tracker agent scheduling environment to a Tivoli Workload Scheduler for z/OS end-to-end scheduling environment with Tivoli Workload Scheduler fault-tolerant agents. We show the benefits of migrating to the fault-tolerant workstations with a step-by-step migration procedure.

17.3.1 Migration benefits

If you plan to migrate to the end solution you can gain the following advantages:

- The use of fault-tolerant technology enables you to continue scheduling without a continuous connection to the z/OS engine.
- Multi-tier architecture enables you to configure your distributed environment into logical and geographic needs through the domain topology.

  The monitoring of workload can be separated based on dedicated distributed views.

  The multi-tier architecture also improves scalability and removes the limitation on the number of tracker agent workstations in Tivoli OPC. (In Tivoli OPC, the designated maximum number of tracker agents was 999, but the practical limit was around 500.)
- High-availability configuration through:
  - The support of AIX High Availability Cluster Multi-Processing (HACMP™), HP Service Guard, and Windows clustering, for example.
  - Support for using host names instead of numeric IP addresses.
  - The ability to change workstation addresses as well as distributed network topology without recycling the Tivoli Workload Scheduler for z/OS Controller. It only requires a plan replan.
- New supported platforms and operating systems, such as:
  - Windows 2000 and Windows XP
  - SUSE Linux Enterprise Server for zSeries Version 7
  - Red Hat Linux (Intel) Version 7.2, 7.3
  - Other third-party access methods such as Tandem

  For a complete list of supported platforms and operating system levels, refer to *IBM Tivoli Workload Scheduler Release Notes Version 8.2* (Maintenance Release April 2004), SC32-1277.
- Support for extended agents.

  Extended agents (XA) are used to extend the job scheduling functions of Tivoli Workload Scheduler to other systems and applications. An extended agent is defined as a workstation that has a host and an access method.

  Extended agents make it possible to run jobs in the end-to-end scheduling solution on:

  - Oracle E-Business Suite
  - PeopleSoft
  - SAP R/3

For more information, refer to *IBM Tivoli Workload Scheduler for Applications User's Guide Version 8.2* (Maintenance Release April 2004), SC32-1278.

- Open *extended agent* interface, which enables you to write extended agents for non-supported platforms and applications. For example, you can write your own extended agent for Tivoli Storage Manager. For more information, refer to *Implementing TWS Extended Agent for Tivoli Storage Manager,* GC24-6030.
- User ID and password definitions for Windows fault-tolerant workstations are easier to implement and maintain. Does not require use of the Tivoli OPC Tracker agent impersonation support.
- IBM Tivoli Business Systems Manager support enables you to integrate the entire end-to-end environment.
- If you use alternate workstations for your tracker agents, be aware that this function is not available in fault-tolerant agents. As part of the fault-tolerant technology, an FTA cannot be an alternate workstation.
- You do not have to touch your planning-related definitions such as run cycles, periods, and calendars.

17.3.2 Migration planning

Before starting the migration process, you may consider the following issues:

- The Job Migration Tool in Tivoli Workload Scheduler for z/OS 8.2 can be used to facilitate the migration from distributed tracker agents to Tivoli Workload Scheduler distributed agents.
- You may choose to not migrate your entire tracker agent environment at once. For better planning, we recommend first deciding which part of your tracker environment is *more eligible* to migrate. This enables you to smoothly migrate to the new fault-tolerant agents. The proper decision can be based on:
  - Agents belonging to a certain business unit
  - Agents running at a specific location or time zone
  - Agents having dependencies to Tivoli Workload Scheduler for z/OS job streams
  - Agents used for testing purposes
- The tracker agents topology is not based on any domain manager structure as used in the Tivoli Workload Scheduler end-to-end solution, so plan the topology configuration that suits your needs.
- Even though you can use centralized scripts to facilitate the migration from distributed tracker agents to Tivoli Workload Scheduler distributed agents, it may be necessary to make some modifications to the JCL (the script) used at

tracker agents when the centralized script for the corresponding fault-tolerant workstation is copied or moved.

For example, this is the case for comments:

– In JCL for tracker agent, a comment line can begin with `//*`

```
//* This is a comment line
```

– In centralized script, a comment line can begin with `//* OPC`

```
//* OPC This is a comment line
```

Tip: We recommend starting the migration with the less critical workload in the environment. The migration process needs some handling and experience; therefore you could start by migrating a test tracker agent with test scripts. If this is successful, you can continue with less critical production job streams and progress to the most important ones.

If centralized script is used, the migration from tracker agents to fault-tolerant workstation should be a simple task. Basically, the migration is done simply by changing the workstation name from the name of a tracker agent workstation to the name of the new fault-tolerant workstation. This is even more true if you follow the migration checklist that is outlined in the following sections.

Also note that with centralized script you can assign a user to a fault-tolerant workstation job exactly the same way as you did for tracker agents (for example, by use of the job submit exit, EQQUX001).

Important: Tivoli OPC tracker agent went out of support on October 31, 2003.

17.3.3 Migration checklist

To guide you through the migration, Table 17-1 provides a step-by-step checklist.

Table 17-1 Migration checklist

Migration actions	Section
1. Install Tivoli Workload Scheduler end-to-end on z/OS mainframe.	“Installing Tivoli Workload Scheduler end-to-end solution” on page 542
2. Install fault-tolerant agents on each tracker agent server or system that should be migrated to end-to-end.	“Installing fault-tolerant agents” on page 543
3. Define the topology for the distributed Tivoli Workload Scheduler network.	“Define the network topology in the end-to-end environment” on page 543

Migration actions	Section
4. Decide whether centralized script, non-centralized script, or a combination of both should be used.	"Decide whether to use centralized or non-centralized scripts" on page 545
5. Define the centralized script.	"Define the centralized script" on page 548
6. Define the non-centralized script.	"Define the non-centralized script" on page 548
7. Define user ID and password for Windows fault-tolerant workstations.	"Define the user and password for Windows FTAs" on page 549
8. Change the workstation name in the job streams from tracker agent workstation name to fault-tolerant workstation name.	"Change the workstation name inside the job streams" on page 549
9. Consider doing some parallel testing before the definitive shift from tracker agents to fault-tolerant agents.	"Parallel testing" on page 550
10. Perform the cutover.	"Perform the cutover" on page 551
11. Educate and train planners and operators.	"Education and training of operators and planners" on page 551

17.3.4 Migration actions

We now explain each step of the migration actions listed in the previous table.

Installing Tivoli Workload Scheduler end-to-end solution

The Tivoli Workload Scheduler for z/OS end-to-end feature is required for the migration, and its installation and configuration are detailed in 15.1, "Installing Tivoli Workload Scheduler for z/OS end-to-end scheduling" on page 380.

Important: Start the installation of the end-to-end solution as early as possible in the migration process to gain as much experience as possible with this new environment before it is handled in the production environment.

End-to-end scheduling is not complicated, but job scheduling on the distributed systems works very differently in an end-to-end environment than in the tracker agent scheduling environment.

Installing fault-tolerant agents

When you have decided which tracker agents to migrate, you can install the Tivoli Workload Scheduler code on the machines or servers that host the tracker agent. This enables you to migrate a mixed environment of tracker agents and fault-tolerant workstations in a more controlled way, because both environments (Tivoli Workload Scheduler and Tivoli OPC Tracker Agents) can coexist on the same physical machine.

Both environments might coexist until you decide to perform the cutover. *Cutover* means switching to the fault-tolerant agent after the testing phase.

Installation of the fault-tolerant agents is explained in detail in 15.2, “Installing FTAs in an end-to-end environment” on page 425.

Define the network topology in the end-to-end environment

In Tivoli Workload Scheduler for z/OS, define the topology of the Tivoli Workload Scheduler network. The definition process contains the following steps:

1. Designing the end-to-end network topology.
2. Definition of the network topology in Tivoli Workload Scheduler for z/OS with the DOMREC and CPUREC keywords.
3. Definition of the fault-tolerant workstations in the Tivoli Workload Scheduler for z/OS database.
4. Activation of the fault-tolerant workstation in the Tivoli Workload Scheduler for z/OS plan by a plan extend or plan replan batchjob.

Tips:

- If you decide to define a topology with domain managers, you should also define backup domain managers.
- To better distinguish the fault-tolerant workstations, follow a consistent naming convention.

After completion of the definition process, each workstation should be defined twice, once for the tracker agent and once for the distributed agent. This way, you can run a distributed agent and a tracker agent on the same computer or server. This should enable you to gradually migrate jobs from tracker agents to distributed agents.

Example: From tracker agent network to end-to-end network

In this example, we illustrate how an existing tracker agent network can be reflected in (or converted to) an end-to-end network topology. This example also

shows the major differences between tracker agent network topology and the end-to-end network topology.

Figure 17-3 shows a classic tracker agent environment. This environment consists of multiple tracker agents on various operating platforms and does not follow any domain topology. All communication with all tracker agents is handled by a single subtask in the Tivoli Workload Scheduler for z/OS Controller started task and does not use any kind of domain structure with multiple levels (tiers) to minimize the load on the Controller.

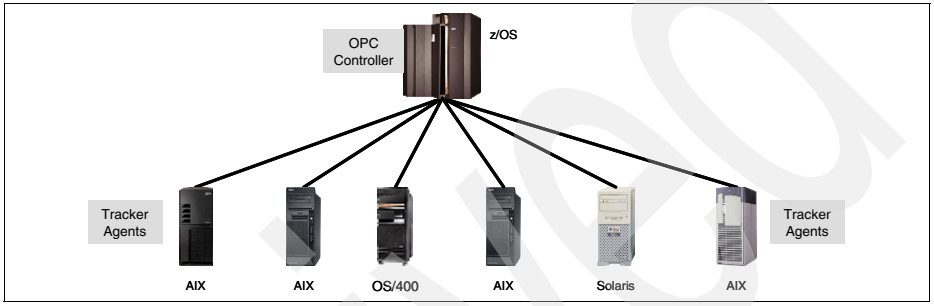


Figure 17-3 A classic tracker agent environment

Figure 17-4 shows how the tracker agent environment in Figure 17-3 can be defined in an end-to-end scheduling environment by use of domain managers, back-up domain managers, and fault-tolerant agents.

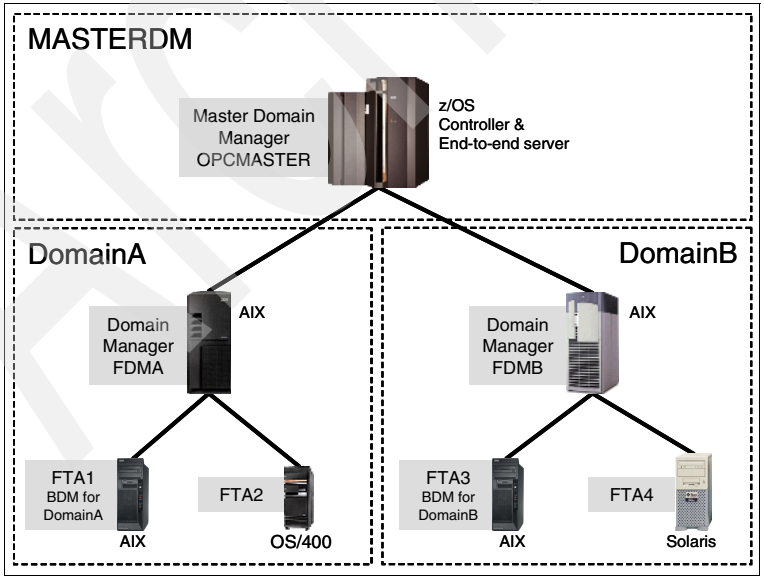


Figure 17-4 End-to-end scheduling network with DMs and FTAs

In the migration phase, it is possible for these two environments to co-exist. *This means that on every machine, a tracker agent and a fault-tolerant workstation are installed.*

Decide whether to use centralized or non-centralized scripts

When migrating from tracker agents to fault-tolerant agents, you have two options regarding scripts: you can use centralized or non-centralized scripts.

If all of the tracker agent scripts (the JCL) are currently stored in the Tivoli Workload Scheduler for z/OS Controller job library, the easiest and most simple solution when migrating to end-to-end will be to use centralized scripts.

But if all of the tracker scripts (the JCL) are placed locally on the tracker agent systems, the easiest and most simple solution when migrating to end-to-end will be to use non-centralized scripts.

Finally, if the tracker agent JCLs are both placed in the Tivoli Workload Scheduler for z/OS Controller job library and locally on the tracker agent system, the easiest will be to migrate to end-to-end scheduling where a combination of centralized and non-centralized script is used.

Use of the Job Migration Tool to help with the migration

This tool can be used to help analyze the existing tracker agent environment to be able to decide whether the tracker agent JCL should be migrated using centralized script, non-centralized script, or a combination.

To run the tool, select option **1.1.5** from the main menu in Tivoli Workload Scheduler for z/OS ISPF. In the panel, enter the name for the tracker agent workstation that you would like to analyze and submit the job generated by Tivoli Workload Scheduler for z/OS.

Note: Before submitting the job, modify it by adding all JOBLIBs for the tracker agent workstation that you are analyzing. Also remember to add JOBLIBs processed by the job-library-read exit (EQQUX002) if it is used.

For a permanent change of the sample job, modify the sample migration job skeleton, EQQWMIGZ.

The tool analyzes the operations (jobs) that are defined on the specified workstation and generates output in four data sets:

1. Report data set (default suffix: LIST)

   Contains warning messages for the processed jobs on the workstation specified as input to the tool. There will be warning messages for:

   – Operations (jobs) that are associated with a job library member that uses JCL variables and directives and that have the centralized script option set to N (No).

   – Scripts (JCL) that do not have variables and are associated with operations that have the centralized script option set to Y (Yes). (This situation lowers performance.)

   – Operations (jobs) for which the tool did not find the JCL (`member not found`) in the JOBLIB libraries specified as input to the tool defined in Tivoli Workload Scheduler.

   Important: Check the tool report for warning messages.

   For jobs (operations) defined with the centralized script option set to No (the default), the tool suggests defining the job on a workstation named `DIST`. For jobs (operations) defined with the centralized script option set to Yes, the tool suggests defining the job on a workstation named `CENT`.

   The last part of the report contains a cross-reference that shows which application (job stream) that the job (operation) is defined in.

   The report is a good starting point for an overview of the migration effort.

   Note: The NT01JOB1 operation (job) is defined in two different applications (job streams): NT01HOUSEKEEPING and NT01TESTAPPL. The NT01JOB1 operation is defined with the centralized script option set to Yes in the NT01TESTAPPL application and No in the NT01HOUSEKEEPING application. That is why the JT01JOB1 is defined on both the CENT and the DIST workstations.

2. JOBLIB data set (default suffix: JOBLIB)

   This library contains a copy of all detected jobs (members) for a specific workstation. The job is copied from the JOBLIB.

   In our example (Example 17-6 on page 547), there are four jobs in this library: NT01AV01, NT01AV02, NT01JOB1, and NT01JOB2.

3. JOBCEN data set (default suffix: JOBCEN)

   This library contains a copy of all jobs (members) that have centralized scripts for a specific workstation that is defined with the centralized script option set to Yes. The job is copied from the JOBLIB.

   In Example 17-6, there are two jobs in this library: NT01JOB1 and NT01JOB2. These jobs were defined in Tivoli Workload Scheduler for z/OS with the centralized script option set to Yes.

4. JOBDIS data set (default suffix: JOBDIS).

   This library contains all jobs (members) that do not have centralized scripts for a specific workstation. These jobs must be transferred to the fault-tolerant workstation.

   In Example 17-6, there are three jobs in this library: NT01AV01, NT01AV02, and NT01JOB1. These jobs were defined in Tivoli Workload Scheduler for z/OS with the centralized script option set to No (the default).

Example 17-6 Report generated by the Job Migration Tool

```
P R I N T O U T   O F   W O R K   S T A T I O N   D E S C R I P T I O N S
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
REPORT TYPE: CROSS-REFERENCE OF JOBNAMES AND ACTIVE APPLICATIONS
================================================================
JOBNAME  APPL ID         VALID TO  OpTYPE_OpNUMBER
-------- --------------- --------  ---------------------------------

NT01AV01 NT01HOUSEKEEPING 31/12/71  DIST_005
         NT01TESTAPPL2    31/12/71  DIST_005
NT01AV02 NT01TESTAPPL2    31/12/71  DIST_010
NT01AV03 NT01TESTAPPL2    31/12/71  DIST_015
  WARNING: NT01AV03 member not found in job library
NT01JOB1 NT01HOUSEKEEPING 31/12/71  DIST_010
         NT01TESTAPPL     31/12/71  CENT_005
  WARNING: Member NT01JOB1 contain directives (//*%OPC) or
  variables (& or % or ?).
Modify the member manually or change the operation(s) type to
centralized.
NT01JOB2 NT01TESTAPPL     31/12/71  CENT_010
  WARNING: You could change operation(s) to NON centralized type.

APPL ID          VALID TO JOBNAME   OpTYPE_OpNUMBER
---------------- -------- --------- ---------------------------------
NT01HOUSEKEEPING 31/12/71 NT01AV01  DIST_005
                          NT01JOB1  DIST_010
NT01TESTAPPL     31/12/71 NT01JOB1  CENT_005
```

```
                          NT01JOB2  CENT_010
NT01TESTAPPL2    31/12/71 NT01AV01  DIST_005
                          NT01AV02  DIST_010
                          NT01AV03  DIST_015
>>>>>>> END OF APPLICATION DESCRIPTION PRINTOUT <<<<<<<
```

Before you migrate the tracker agent to a distributed agent, you should use this tool to obtain these files for help in deciding whether the jobs should be defined with centralized or decentralized scripts.

Define the centralized script

If you decide to use centralized scripts for all or some of the tracker agent jobs:

1. Run the job migration tool for each tracker agent workstation and analyze the generated report.
2. Change the value of the centralized script flag to `Yes`, based on the result of the job migration tool output and your decision.
3. Run the job migration tool as many times as you want. For example, you can run until there are no warning messages and all jobs are defined on the correct workstation in the report (the CENT workstation).
4. Change the generated JCL (jobs) in the JOBCEN data set (created by the migration tool); for example, it could be necessary to change the comments line from **//*** to `//* OPC`.

   Note: If you plan to run the migration tool several times, you should copy the job to another library when it has been changed and is ready for the switch to avoid it being replaced by a new run of the migration tool.

5. The copied and amended members (jobs) can be activated one by one when the corresponding operation in the Tivoli Workload Scheduler for z/OS application is changed from the tracker agent workstation to the fault-tolerant workstation.

Define the non-centralized script

If you decide to use non-centralized scripts for all or some of the tracker agent jobs:

1. Run the job migration tool for each tracker agent workstation and analyze the generated report.
2. Run the job migration tool as many times as you want. For example, you can run until there are no warning messages and all jobs are defined on the correct workstation in the report (the DIST workstation).

3. Transfer the scripts from the JOBDIS data set (created by the migration tool) to the distributed agents.
4. Create a member in the script library (SCRPTLIB/EQQSCLIB) for every job in the JOBDIS data set and, optionally, for the jobs in JOBCEN (if you decide to change these jobs from use of centralized script to non-centralized script).

> **Note:** The job submit exit EQQUX001 is not called for non-centralized script jobs.

Define the user and password for Windows FTAs

For each user running jobs on Windows fault-tolerant agents, define a new USRREC statement to provide the Windows user and password. USRREC is defined in the member of the EQQPARM library as specified by the USERMEM keyword in the TOPOLOGY statement.

> **Important:** Because the passwords are not encrypted, we strongly recommend that you protect the data set containing the USRREC definitions with your security product.

If you use the impersonation support for NT tracker agent workstations, it does not interfere with the USRREC definitions. The impersonation support assigns a user ID based on the user ID from exit EQQUX001. Because the exit is not called for jobs with non-centralized script, impersonation support is not used any more.

Change the workstation name inside the job streams

At this point in the migration, the end-to-end scheduling environment should be active, and the fault-tolerant workstations on the systems with tracker agents should be active and linked in the plan in Tivoli Workload Scheduler for z/OS.

The plan in Tivoli Workload Scheduler for z/OS and the Symphony file on the fault-tolerant agents does not contain any job streams with jobs that are scheduled on the tracker agent workstations.

The job streams (applications) in the Tivoli Workload Scheduler for z/OS Controller are still pointing to the tracker agent workstations.

In order to submit workload to the distributed environment, you must change the workstation name of your existing job definitions to the new FTA, or define new job streams to replace the job streams with the old tracker agent jobs.

Notes:

- It is not possible to change the workstation within a job instance from a tracker agent to a fault-tolerant workstation via the Job Scheduling Console. We have already addressed this issue of development. The change can be performed via the GUI (ISPF) and the batch loader program.
- Changes to the workstation affect only the job stream database. If you want to take this modification into the plans, you must run a long term-plan (LTP) *modify all* batch job and current plan extend or replan batch job.
- The highest acceptable return code for operations on fault-tolerant workstations is 0. If you have a tracker agent operation with highest return code set to 8 and you change the workstation for this operation from a tracker agent workstation to a fault-tolerant workstation, you will not be able to save the modified application.

  When trying to save the application you will see error message:

  ```
  EQQA531E Inconsistent option when FT work station
  ```

  Be aware of this if you are planning to use Tivoli Workload Scheduler for z/OS mass update functions or unload/reload functions to update a large number of applications.

Parallel testing

If possible, do some parallel testing before the cutover. With parallel testing, you run the same job flow on both types of workstations: tracker agent workstations and fault-tolerant workstations.

The only problem with parallel testing is that it requires duplicate versions of the applications (job streams): one application for the tracker agent and one application for the fault-tolerant workstation. Also, you cannot run the same job in both applications, so one of the jobs must be changed to a dummy job.

Some initial setup is required to do parallel testing, but when done it will be possible to verify that the jobs are executed in same sequence and that operators and planners can gain some experience with the new environment.

Another approach could be to migrate a few applications from tracker agents to fault-tolerant agents and use these applications to verify the migration strategy and migrated jobs (JCL/script), and gain some experience. When satisfied with the test result of these applications, you can migrate the rest of the applications.

Perform the cutover

When the parallel testing has been completed with satisfactory results, you can do the final cutover. For example, the process can be:

- Change all workstation names from tracker agent workstation to fault-tolerant workstation for all operations in the Tivoli Workload Scheduler for z/OS Controller.

  This can be done with the Tivoli Workload Scheduler for z/OS mass update function or by the unload (with the Batch Command Interface Tool) edit, and batchload (with Tivoli Workload Scheduler for z/OS batchloader) process.

- Run the *Extend of long-term plan* batch job or *Modify All of long-term plan* in Tivoli Workload Scheduler for z/OS.

  Verify that the changed applications and operations look correct in the long-term plan.

- Run the *Extend of plan* (current plan) batch job.
  - Verify that the changed applications and operations look correct in the plan.
  - Verify that the tracker agent jobs have been moved to the new fault-tolerant workstations and that there are no jobs on the tracker agent workstations.

Education and training of operators and planners

Tracker agents and fault-tolerant workstations work differently, and there are new options related to jobs on fault-tolerant workstations. Handling of fault-tolerant workstations is different from handling for tracker agents.

A tracker agent workstation can be set to Active or Offline and can be defined with open intervals and servers. A fault-tolerant workstation can be started, stopped, linked, or unlinked.

To ensure that the migration from tracker agents to fault-tolerant workstations will be successful, be sure to plan for education of your planners and operators.

17.3.5 Migrating backward

Normally, it should not be necessary to migrate backward because it is possible to run the two environments in parallel. As we have shown, you can run a tracker agent and a fault-tolerant agent on the same physical machine. If the preparation, planning, and testing of the migration is done as described in the previous chapters, it should not be necessary to migrate backward.

If a situation forces backward migration from the fault-tolerant workstations to tracker agents, follow these steps:

1. Install the tracker agent on the computer. (This is necessary only if you have uninstalled the tracker agent.)
2. Define a new destination in the ROUTOPTS initialization statement of the Controller and restart the Controller.
3. Make a duplicate of the workstation definition of the computer. Define the new workstation as `Computer Automatic` instead of Fault Tolerant and specify the destination that you defined in step 2. This way, the same computer can be run as a fault-tolerant workstation and as a tracker agent for smoother migration.
4. For non-centralized scripts, copy the script from the fault-tolerant workstation repository to the JOBLIB. As an alternative, copy the script to a local directory that can be accessed by the tracker agent and create a JOBLIB member to execute the script. You can use FTP for this.
5. Implement the EQQUX001 sample to execute jobs with the correct user ID.
6. Modify the workstation name inside the operation. Remember to change the JOBNAME if the member in the JOBLIB has a name different from the member of the script library.

17.4 Conversion from Tivoli Workload Scheduler network to Tivoli Workload Scheduler for z/OS managed network

In this section, we outline the guidelines for converting a Tivoli Workload Scheduler network to a Tivoli Workload Scheduler for z/OS managed network.

The distributed Tivoli Workload Scheduler network is managed by a Tivoli Workload Scheduler master domain manager, which manages the databases and the plan. Converting the Tivoli Workload Scheduler managed network to a Tivoli Workload Scheduler for z/OS managed network means that responsibility for database and plan management move from the Tivoli Workload Scheduler master domain manager to the Tivoli Workload Scheduler for z/OS engine.

17.4.1 Illustration of the conversion process

Figure 17-5 shows a distributed Tivoli Workload Scheduler network. The database management and daily planning are carried out by the Tivoli Workload Scheduler master domain manager.

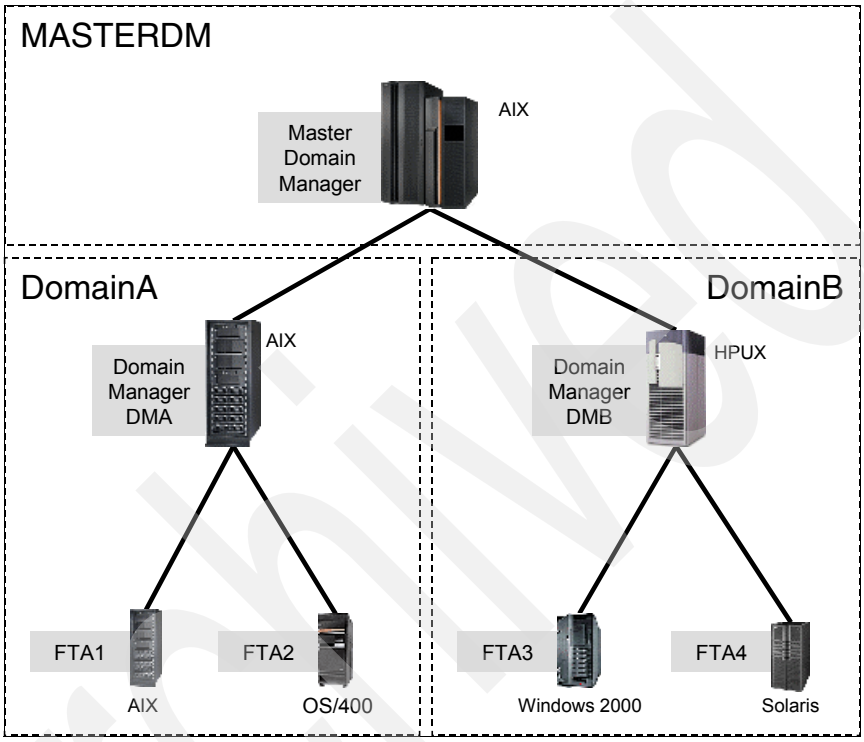


Figure 17-5 Tivoli Workload Scheduler distributed network with a master domain manager

Figure 17-6 shows a Tivoli Workload Scheduler for z/OS managed network. Database management and daily planning are carried out by the Tivoli Workload Scheduler for z/OS engine.

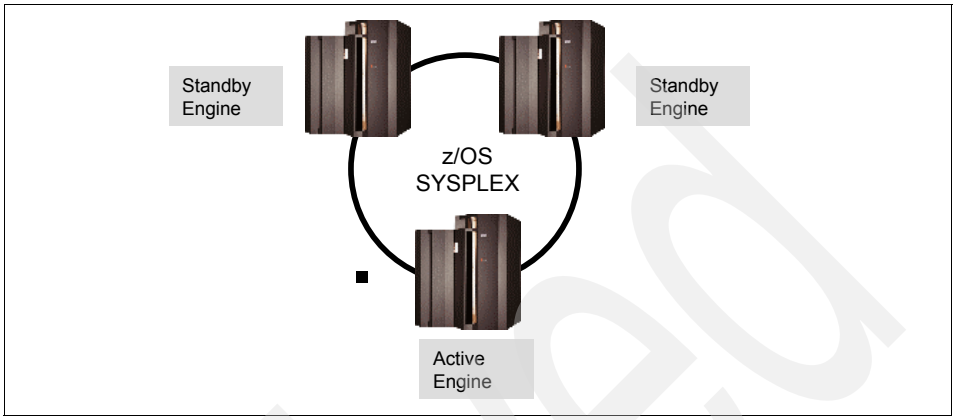


Figure 17-6 Tivoli Workload Scheduler for z/OS network

The conversion process changes the Tivoli Workload Scheduler master domain manager to the first-level domain manager and then connect it to the Tivoli Workload Scheduler for z/OS engine (new master domain manager). The result of the conversion is a new end-to-end network managed by the Tivoli Workload Scheduler for z/OS engine (Figure 17-7).

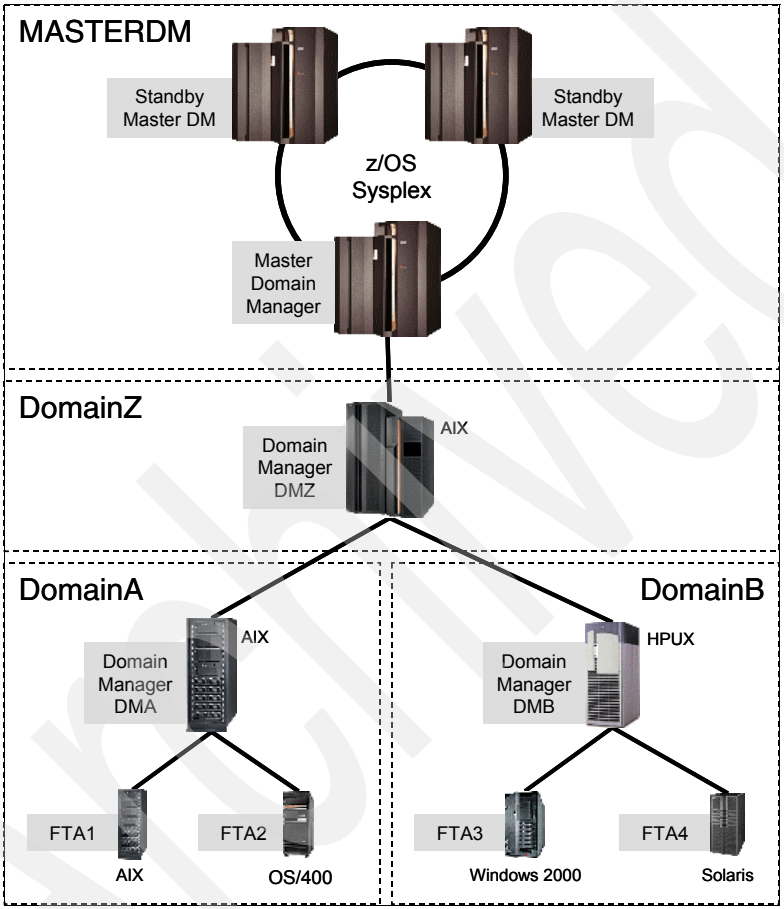


Figure 17-7 Tivoli Workload Scheduler for z/OS managed end-to-end network

17.4.2 Considerations before doing the conversion

Before you start to convert your Tivoli Workload Scheduler managed network to a Tivoli Workload Scheduler for z/OS managed network, you should evaluate the positives and negatives of doing the conversion.

The pros and cons of doing the conversion will differ from installation to installation. Some installations will gain significant benefits from conversion, and

other installations will gain fewer benefits. Based on the outcome of the evaluation of pros and cons, it should be possible to make the right decision for your specific installation and current usage of Tivoli Workload Scheduler as well as Tivoli Workload Scheduler for z/OS.

Some important aspects of the conversion that you should consider are:

- How is your Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS organization today?
  - Do you have two independent organizations working independently of each other?
  - Do you have two groups of operators and planners to manage Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS?
  - Or do you have one group of operators and planners that manages both the Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS environments?
  - Do you use considerable resources keeping a high skill level for both products, Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS?
- How integrated is the workload managed by Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS?
  - Do you have dependencies between jobs in Tivoli Workload Scheduler and in Tivoli Workload Scheduler for z/OS?
  - Or do most of the jobs in one workload scheduler run independent of jobs in the other scheduler?
  - Have you already managed to solve dependencies between jobs in Tivoli Workload Scheduler and in Tivoli Workload Scheduler for z/OS efficiently?
- The current use of Tivoli Workload Scheduler–specific functions are not available in Tivoli Workload Scheduler for z/OS.
  - How intensive is the use of prompts, file dependencies, and “repeat range” (run job every 10 minutes) in Tivoli Workload Scheduler?

    Can these Tivoli Workload Scheduler–specific functions be replaced by Tivoli Workload Scheduler for z/OS–specific functions or should they be handled in another way?

    Does it require some locally developed tools, programs, or workarounds?
  - How extensive is the use of Tivoli Workload Scheduler job recovery definitions?

    Is it possible to handle these Tivoli Workload Scheduler recovery definitions in another way when the job is managed by Tivoli Workload Scheduler for z/OS?

Does it require some locally developed tools, programs, or workarounds?

- Will Tivoli Workload Scheduler for z/OS give you some of the functions you are missing in Tivoli Workload Scheduler today?
  - Extended planning capabilities, long-term plan, current plan that spans more than 24 hours?
  - Better handling of carry-forward job streams?
  - Powerful run-cycle and calendar functions?
- Which platforms or systems are going to be managed by the Tivoli Workload Scheduler for z/OS end-to-end scheduling?
- What kind of integration do you have between Tivoli Workload Scheduler and, for example, SAP R/3, PeopleSoft, or Oracle applications?
- Partial conversion of some jobs from the Tivoli Workload Scheduler-managed network to the Tivoli Workload Scheduler for z/OS managed network?

  Partial conversion: About 15% of your Tivoli Workload Scheduler managed jobs or workload is directly related to the Tivoli Workload Scheduler for z/OS jobs or workload. This means that the Tivoli Workload Scheduler jobs are either predecessors or successors to Tivoli Workload Scheduler for z/OS jobs. The current handling of these interdependencies is not effective or stable with your current solution.

  Converting the 15% of jobs to Tivoli Workload Scheduler for z/OS managed scheduling using the end-to-end solution will stabilize dependency handling and make scheduling more reliable. Note that this requires two instances of Tivoli Workload Scheduler workstations (one each for Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS).

- How much effort is involved to convert Tivoli Workload Scheduler database object definitions to Tivoli Workload Scheduler for z/OS database object definitions?

  Will it be possible to convert the database objects with reasonable resources and within a reasonable time frame?

17.4.3 Conversion process from Tivoli Workload Scheduler to Tivoli Workload Scheduler for z/OS

The process of converting from a Tivoli Workload Scheduler-managed network to a Tivoli Workload Scheduler for z/OS-managed network has several steps. In the following description, we assume that we have an active Tivoli Workload Scheduler for z/OS environment as well as an active Tivoli Workload Scheduler

environment. We also assume that the Tivoli Workload Scheduler for z/OS end-to-end server is installed and ready for use. The conversion process mainly contains the following steps or tasks:

1. Plan the conversion and establish new naming standards.
2. Install new Tivoli Workload Scheduler workstation instances dedicated to communicating with the Tivoli Workload Scheduler for z/OS server.
3. Define the topology of the Tivoli Workload Scheduler network in Tivoli Workload Scheduler for z/OS and define associated Tivoli Workload Scheduler for z/OS fault-tolerant workstations.
4. Create JOBSCR members (in the SCRPTLIB data set) for all Tivoli Workload Scheduler–managed jobs that should be converted.
5. Convert the database objects from Tivoli Workload Scheduler format to Tivoli Workload Scheduler for z/OS format.
6. Educate planners and operators in the new Tivoli Workload Scheduler for z/OS server functions.
7. Test and verify the conversion and finalize for production.

The sequencing of these steps may be different in your environment, depending on the strategy that you will follow when doing your own conversion.

Step 1. Planning the conversion

The conversion from Tivoli Workload Scheduler-managed scheduling to Tivoli Workload Scheduler for z/OS-managed scheduling can be a major project that requires several resources. It depends on the current size and usage of the Tivoli Workload Scheduler environment. Planning of the conversion is an important task, and it can be used to estimate the effort required to do the conversion as well as detail the different conversion steps.

In the planning phase you should try to identify special usage of Tivoli Workload Scheduler functions or facilities that are not easily converted to Tivoli Workload Scheduler for z/OS. Furthermore, you should try to outline how these functions or facilities should be handled when scheduling is done by Tivoli Workload Scheduler for z/OS.

Part of planning is also establishing the new naming standards for all or some of the Tivoli Workload Scheduler objects that are going to be converted. Some examples:

- Naming standards for the fault-tolerant workstations in Tivoli Workload Scheduler for z/OS

  Names for workstations can be up to 16 characters in Tivoli Workload Scheduler (if you are using expanded databases). In Tivoli Workload

Scheduler for z/OS, workstation names can be up to four characters. This means that you have to establish a new naming standard for the fault-tolerant workstations in Tivoli Workload Scheduler for z/OS.

- Naming standards for job names

  In Tivoli Workload Scheduler you can specify job names with lengths of up to 40 characters (if you are using expanded databases). In Tivoli Workload Scheduler for z/OS, job names can be up to eight characters. This means that you have to establish a new naming standard for jobs on fault-tolerant workstations in Tivoli Workload Scheduler for z/OS.

- Adoption of the existing Tivoli Workload Scheduler for z/OS object naming standards

  You probably already have naming standards for job streams, workstations, job names, resources, and calendars in Tivoli Workload Scheduler for z/OS. When converting Tivoli Workload Scheduler database objects to the Tivoli Workload Scheduler for z/OS databases, you must adopt the Tivoli Workload Scheduler for z/OS naming standard.

- Access to the objects in Tivoli Workload Scheduler for z/OS database and plan

  Access to Tivoli Workload Scheduler for z/OS databases and plan objects are protected by your security product (for example, RACF). Depending on the naming standards for the imported Tivoli Workload Scheduler objects, you may need to modify the definitions in your security product.

- Is the current Tivoli Workload Scheduler network topology suitable and can it be implemented directly in a Tivoli Workload Scheduler for z/OS server?

  The current Tivoli Workload Scheduler network topology might need some adjustments as it is implemented today to be optimal. If your Tivoli Workload Scheduler network topology is not optimal, it should be reconfigured when implemented in Tivoli Workload Scheduler for z/OS end-to-end.

Step 2. Install Tivoli Workload Scheduler workstation instances for Tivoli Workload Scheduler for z/OS

With Tivoli Workload Scheduler workstation instances, we mean installation and configuration of a new Tivoli Workload Scheduler engine. This engine should be configured to be a domain manager, fault-tolerant agent, or a backup domain manager, according to the Tivoli Workload Scheduler production environment you are going to mirror. Following this approach, you will have two instances on all the Tivoli Workload Scheduler managed systems:

1. One old Tivoli Workload Scheduler workstation instance dedicated to the Tivoli Workload Scheduler master.

2. One new Tivoli Workload Scheduler workstation instance dedicated to the Tivoli Workload Scheduler for z/OS engine (master). Remember to use different port numbers.

By creating dedicated Tivoli Workload Scheduler workstation instances for Tivoli Workload Scheduler for z/OS scheduling, you can start testing the new environment without disturbing the distributed Tivoli Workload Scheduler production environment. This also makes it possible to do partial conversion, testing, and verification without interfering with the Tivoli Workload Scheduler production environment.

You can chose different approaches for the conversion:

- Try to group your Tivoli Workload Scheduler job streams and jobs into logical and isolated groups and then convert them, group by group.
- Convert all job streams and jobs, run some parallel testing and verification, and then do the switch from Tivoli Workload Scheduler–managed scheduling to Tivoli Workload Scheduler for z/OS–managed scheduling in one final step.

The suitable approach differs from installation to installation. Some installations will be able to group job streams and jobs into isolated groups; others will not. You have to decide the strategy for the conversion based on your installation.

Note: If you decide to reuse the Tivoli Workload Scheduler distributed workstation instances in your Tivoli Workload Scheduler for z/OS–managed network, this is also possible. You may decide to move the distributed workstations one by one (depending on how you have grouped your job streams and how you are doing the conversion). When a workstation is going to be moved to Tivoli Workload Scheduler for z/OS, you simply change the port number in the localopts file on the Tivoli Workload Scheduler workstation. The workstation will then be active in Tivoli Workload Scheduler for z/OS at the next plan extension, replan, or redistribution of the Symphony file. (Remember to create the associated DOMREC and CPUREC definitions in the Tivoli Workload Scheduler for z/OS initialization statements.)

Step 3. Define topology of Tivoli Workload Scheduler network in Tivoli Workload Scheduler for z/OS

The topology for your Tivoli Workload Scheduler distributed network can be implemented directly in Tivoli Workload Scheduler for z/OS. This is done by creating the associated DOMREC and CPUREC definitions in the Tivoli Workload Scheduler for z/OS initialization statements.

To activate the topology definitions, create the associated definitions for fault-tolerant workstations in the Tivoli Workload Scheduler for z/OS workstation

database. Tivoli Workload Scheduler for z/OS extend or replan will activate these new workstation definitions.

If you are using a dedicated Tivoli Workload Scheduler workstation for Tivoli Workload Scheduler for z/OS scheduling, you can create the topology definitions in an early stage of the conversion process. This way you can:

- Verify that the topology definitions are correct in Tivoli Workload Scheduler for z/OS.
- Verify that the dedicated fault-tolerant workstations are linked and available.
- Start getting some experience with the management of fault-tolerant workstations and a distributed Tivoli Workload Scheduler network.
- Implement monitoring and handling routines in your automation application on z/OS.

Step 4. Create JOBSCR members for all Tivoli Workload Scheduler–managed jobs

Tivoli Workload Scheduler–managed jobs that should be converted to Tivoli Workload Scheduler for z/OS must be defined in the SCRPTLIB data set. For every active job defined in the Tivoli Workload Scheduler database, you define a member in the SCRPTLIB data set containing:

- Name of the script or command for the job (defined in the JOBREC JOBSCRS() or the JOBREC JOBCMD specification)
- Name of the user ID that the job should execute under (defined in the JOBREC JOBUSR() specification)

Note: If the same job script is going to be executed on several systems (it is defined on several workstations in Tivoli Workload Scheduler), you only have to create one member in the SCRPTLIB data set. This job (member) can be defined on several fault-tolerant workstations in several job streams in Tivoli Workload Scheduler for z/OS. It requires that the script is placed in a common directory (path) across all systems.

Step 5. Convert database objects from Tivoli Workload Scheduler to Tivoli Workload Scheduler for z/OS

Tivoli Workload Scheduler database objects that should be converted (job streams, resources, and calendars) probably cannot be converted directly to Tivoli Workload Scheduler for z/OS. In this case you must amend the Tivoli Workload Scheduler database objects to Tivoli Workload Scheduler for z/OS

format and create the corresponding objects in the respective Tivoli Workload Scheduler for z/OS databases.

Pay special attention to object definitions such as:

- Job stream run-cycles for job streams and use of calendars in Tivoli Workload Scheduler
- Use of local (workstation-specific) resources in Tivoli Workload Scheduler (local resources converted to global resources by the Tivoli Workload Scheduler for z/OS master)
- Jobs defined with “repeat range” (for example, run every 10 minutes in job streams)
- Job streams defined with dependencies on job stream level
- Jobs defined with Tivoli Workload Scheduler recovery actions

For these object definitions, you have to design alternative ways of handling for Tivoli Workload Scheduler for z/OS.

Step 6. Education for planners and operators

Some of the handling of distributed Tivoli Workload Scheduler jobs in Tivoli Workload Scheduler for z/OS will be different from the handling in Tivoli Workload Scheduler. Also, some specific fault-tolerant workstation features will be available in Tivoli Workload Scheduler for z/OS.

You should plan for the education of your operators and planners so that they have knowledge of:

- How to define jobs and job streams for the Tivoli Workload Scheduler fault-tolerant workstations
- Specific rules to be followed for scheduling objects related to fault-tolerant workstations
- How to handle jobs and job streams on fault-tolerant workstations
- How to handle resources for fault-tolerant workstations
- The implications of doing, for example, Symphony redistribution
- How Tivoli Workload Scheduler for z/OS end-to-end scheduling works (engine, server, domain managers)
- How the Tivoli Workload Scheduler network topology has been adopted in Tivoli Workload Scheduler for z/OS

Step 7. Test and verify conversion and finalize for production

After testing your approach for the conversion, doing some trial conversions, and testing the conversion carefully, it is time to do the final conversion to Tivoli Workload Scheduler for z/OS.

The goal is to reach this final conversion and switch from Tivoli Workload Scheduler scheduling to Tivoli Workload Scheduler for z/OS scheduling within a reasonable time frame and with a reasonable level of errors. If the period when you are running Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS will be too long, your planners and operators must handle two environments in this period. This is not effective and can cause some frustration for both planners and operators.

The key to a successful conversion is good planning, testing, and verification. When you are comfortable with the testing and verification it is safe to do the final conversion and finalize for production.

Tivoli Workload Scheduler for z/OS will then handle the central and the distributed workload, and you will have one focal point for your workload. The converted Tivoli Workload Scheduler production environment can be stopped.

17.4.4 Some guidelines to automate the conversion process

If you have a large Tivoli Workload Scheduler scheduling environment, manual conversion will be too time-consuming. In this case you should consider trying to automate some or all of the conversion from Tivoli Workload Scheduler to Tivoli Workload Scheduler for z/OS.

One obvious place to automate is the conversion of Tivoli Workload Scheduler database objects to Tivoli Workload Scheduler for z/OS database objects. Although this is not a trivial task, some automation can be implemented. Automation requires some locally developed tools or programs to handle conversion of the database objects.

Guidelines to help with automating the conversion process include:

- Create text copies of all Tivoli Workload Scheduler database objects by using the **composer create** command (Example 17-7).

Example 17-7 Tivoli Workload Scheduler objects creation

```
composer create calendars.txt    from CALENDARS
composer create workstations.txt from CPU=@
composer create jobdef.txt       from JOBS=@#@
composer create jobstream.txt    from SCHED=@#@
composer create parameter.txt    from PARMS
```

```
composer create resources.txt    from RESOURCES
composer create prompts.txt      from PROMPTS
composer create users.txt        from USERS=@#@
```

These text files are a good starting point when trying to estimate the effort and time for conversion from Tivoli Workload Scheduler to Tivoli Workload Scheduler for z/OS.

- Use the workstations.txt file when creating the topology definitions (DOMREC and CPUREC) in Tivoli Workload Scheduler for z/OS.

  Creating the topology definitions in Tivoli Workload Scheduler for z/OS based on the workstations.txt file is quite straightforward. The task can be automated coding using a program (script or REXX) that reads the worksations.txt file and converts the definitions to DOMREC and CPUREC specifications.

  Restriction: Tivoli Workload Scheduler CPU class definitions cannot be converted directly to similar definitions in Tivoli Workload Scheduler for z/OS.

- Use the jobdef.txt file when creating the SCRPTLIB members.

  jobdef.txt has the workstation name for the script (used in the job stream definition), the script name (goes to the JOBREC JOBSCR() definition), the stream logon (goes to the JOBREC JOBUSR() definition), the description (can be added as comments in the SCRPTLIB member), and the recovery definition.

  The recovery definition needs special consideration because it cannot be converted to Tivoli Workload Scheduler for z/OS auto-recovery. Here you need to make some workarounds. Use of Tivoli Workload Scheduler CPU class definitions needs special consideration. The job definitions using CPU classes probably have to be copied to separate workstation-specific job definitions in Tivoli Workload Scheduler for z/OS. The task can be automated coding using a program (scripts or REXX) that reads the jobdef.txt file and converts each job definition to a member in the SCRPTLIB. If you have many Tivoli Workload Scheduler job definitions, having a program that can help automate this task can save a considerable amount of time.

- The users.txt file (if you have Windows NT/2000 jobs) is converted to USRREC initialization statements on Tivoli Workload Scheduler for z/OS.

  Be aware that the password for the user IDs is encrypted in the users.txt file, so you cannot automate the conversion right away. You must get the password as it is defined on the Windows workstations and type it in the USRREC USRPSW() definition.

- The jobstream.txt file is used to generate corresponding job streams in Tivoli Workload Scheduler for z/OS. The calendars.txt file is used in connection with the jobstream.txt file when generating run cycles for the job streams in Tivoli Workload Scheduler for z/OS. It could be necessary to create additional calendars in Tivoli Workload Scheduler for z/OS.

  When doing the conversion, you should notice that:

  - Some of the Tivoli Workload Scheduler job stream definitions cannot be converted directly to Tivoli Workload Scheduler for z/OS job stream definitions (for example: prompts, workstation-specific resources, file dependencies, and jobs with repeat range).

    For these definitions, you must analyze the usage and find other ways to implement similar functions when using Tivoli Workload Scheduler for z/OS.

  - Some of the Tivoli Workload Scheduler job stream definitions must be amended to Tivoli Workload Scheduler for z/OS definitions. For example:

    - Dependencies on job stream level (use dummy start and end jobs in Tivoli Workload Scheduler for z/OS for job stream dependencies).

      Note that dependencies are also dependencies to prompts, file dependencies, and resources.

    - Tivoli Workload Scheduler job and job stream priority (0 to 101) must be amended to Tivoli Workload Scheduler for z/OS priority (1 to 9). Furthermore, priority in Tivoli Workload Scheduler for z/OS is always on job stream level. (It is not possible to specify priority on job level.)

    - Job stream run cycles (and calendars) must be converted to Tivoli Workload Scheduler for z/OS run cycles (and calendars).

  - Description texts longer than 24 characters are not allowed for job streams or jobs in Tivoli Workload Scheduler for z/OS. If you have Tivoli Workload Scheduler job streams or jobs with more than 24 characters of description text, you should consider adding this text as Tivoli Workload Scheduler for z/OS operator instructions.

  If you have a large number of Tivoli Workload Scheduler job streams, manual handling of job streams can be too time-consuming. The task can be automated to a certain extend coding program (script or REXX).

  A good starting point is to code a program that identifies all areas where you need special consideration or action. Use the output from this program to estimate the effort of doing the conversion. The output can also be used to identify and group used Tivoli Workload Scheduler functions where special workarounds must be performed when converting to Tivoli Workload Scheduler for z/OS.

The program can be further refined to handle the actual conversion, performing the following steps:

- Read all of the text files.
- Analyze the job stream and job definitions.
- Create corresponding Tivoli Workload Scheduler for z/OS job streams with amended run cycles and jobs.
- Generate a file with Tivoli Workload Scheduler for z/OS batch loader statements for job streams and jobs (batch loader statements are Tivoli Workload Scheduler for z/OS job stream definitions in a format that can be loaded directly into the Tivoli Workload Scheduler for z/OS databases).

The batch loader file can be sent to the z/OS system and used as input to the Tivoli Workload Scheduler for z/OS batch loader program. The Tivoli Workload Scheduler for z/OS batch loader will read the file (data set) and create the job streams and jobs defined in the batch loader statements.

- The resources.txt file is used to define the corresponding resources in Tivoli Workload Scheduler for z/OS.

  Remember that local (workstation-specific) resources are not allowed in Tivoli Workload Scheduler for z/OS. This means that the Tivoli Workload Scheduler workstation-specific resources will be converted to *global* special resources in Tivoli Workload Scheduler for z/OS.

  The Tivoli Workload Scheduler for z/OS engine is directly involved when resolving a dependency to a global resource. A fault-tolerant workstation job must interact with the Tivoli Workload Scheduler for z/OS engine to resolve a resource dependency. This can jeopardize the fault tolerance in your network.

- The use of parameters in the parameters.txt file must be analyzed.

  What are the parameters used for?

  - Are the parameters used for date calculations?
  - Are the parameters used to pass information from one job to another job (using the Tivoli Workload Scheduler **parms** command)?
  - Are the parameters used as parts of job definitions (for example, to specify where the script is placed)?

  Depending on how you used the Tivoli Workload Scheduler parameters, there will be different approaches when converting to Tivoli Workload Scheduler for z/OS. Unless you use parameters as part of Tivoli Workload Scheduler object definitions, you usually do not have to do any conversion. Parameters will still work after the conversion.

  You have to copy the parameter database to the home directory of the Tivoli Workload Scheduler fault-tolerant workstations. The **parms** command can still

be used locally on the fault-tolerant workstation when managed by Tivoli Workload Scheduler for z/OS.

We will show how to use Tivoli Workload Scheduler parameters in connection with Tivoli Workload Scheduler for z/OS JCL variables. This is a way to pass values for Tivoli Workload Scheduler for z/OS JCL variables to Tivoli Workload Scheduler parameters so that they can be used locally on the fault-tolerant workstation.

17.5 Tivoli Workload Scheduler for z/OS end-to-end fail-over scenarios

In this section, we describe how to make the Tivoli Workload Scheduler for z/OS end-to-end environment fail-safe and plan for system outages. We also show some fail-over scenario examples.

To make your Tivoli Workload Scheduler for z/OS end-to-end environment fail-safe, you have to:

- Configure Tivoli Workload Scheduler for z/OS backup engines (also called hot standby engines) in your sysplex.

  If you do not run sysplex, but have more than one z/OS system with shared DASD, then you should make sure that the Tivoli Workload Scheduler for z/OS engine can be moved from one system to another without any problems.
- Configure your z/OS systems to use a virtual IP address (VIPA).

  VIPA is used to make sure that the Tivoli Workload Scheduler for z/OS end-to-end server always gets the same IP address no matter which z/OS system it is run on. VIPA assigns a system-independent IP address to the Tivoli Workload Scheduler for z/OS server task.

  If using VIPA is not an option, you should consider other ways of assigning a system-independent IP address to the Tivoli Workload Scheduler for z/OS server task. For example, this can be a hostname file, DNS, or stack affinity.
- Configure a backup domain manager for the first-level domain manager.

Refer to the Tivoli Workload Scheduler for z/OS end-to-end configuration, shown in Figure 17-1 on page 530, for the fail-over scenarios.

When the environment is configured to be fail-safe, the next step is to test that the environment actually *is* fail-safe. We did the following fail-over tests:

- Switch to the Tivoli Workload Scheduler for z/OS backup engine
- Switch to the Tivoli Workload Scheduler backup domain manager

17.5.1 Configure Tivoli Workload Scheduler for z/OS backup engines

To ensure that the Tivoli Workload Scheduler for z/OS engine will be started, either as an active engine or standby engine, we specify:

```
OPCOPTS  OPCHOST(PLEX)
```

In the initialization statements for the Tivoli Workload Scheduler for z/OS engine (pointed to by the member of the EQQPARM library as specified by the parm parameter on the JCL EXEC statement), OPCHOST(PLEX) means that the engine has to start as the controlling system. If there already is an active engine in the XCF group, the startup for the engine continues on standby engine.

Note: `OPCOPTS OPCHOST(YES)` must be specified if you start the engine with an empty checkpoint data set. This could be the case the first time you start a newly installed engine or after you have migrated from a previous release of Tivoli Workload Scheduler for z/OS.

OPCHOST(PLEX) is valid only when the n XCF group and member have been specified. Also, this selection requires that Tivoli Workload Scheduler for z/OS is running on a z/OS/ESA Version 4 Release 1 or later. Because we are running z/OS 1.3, we can use the OPCHOST PLEX(YES) definition. We specify Example 17-8 in the XCF group and member definitions for the engine.

Example 17-8 XCF group and member definitions

```
XCFOPTS  GROUP(TWS820)
         MEMBER(TWSC&SYSNAME.)
/*       TAKEOVER(SYSFAIL,HOSTFAIL)    Do takeover manually !!
*/
```

Tip: We use the z/OS sysplex-wide SYSNAME variable when specifying the member name for the engine in the sysplex. Using z/OS variables this way, we can have common Tivoli Workload Scheduler for z/OS parameter member definitions for all our engines (and agents as well).

For example, when the engine is started on SC63, the MEMBER(TWSC&SYSNAME) will be MEMBER(TWSCSC63).

You must have unique member names for all your engines (active and standby) running in the same sysplex. We assure this by using the sysname variable.

Tip: We have not activated the TAKEOVER(SYSFAIL,HOSTFAIL) parameter in XCFOPTS because we do not want the engine to switch automatically to one of its backup engines in case the active engine fails or the system fails.

Because we have not specified the TAKEOVER parameter, we are making the switch to one of the backup engines manually. The switch is made by issuing the following modify command on the z/OS system where you want the backup engine to take over:

```
F TWSC,TAKEOVER
```

In this example, `TWSC` is the name of our Tivoli Workload Scheduler for z/OS backup engine started task (same name on all systems in the sysplex).

The takeover can be managed by SA/390, for example. This way SA/390 can integrate the switch to a backup engine with other automation tasks in the engine or on the system.

We did not define a Tivoli Workload Scheduler for z/OS APPC server task for the Tivoli Workload Scheduler for z/OS panels and PIF programs, but it is strongly recommended that you use a Tivoli Workload Scheduler for z/OS APPC server task in sysplex environments where the engine can be moved to different systems in the sysplex. If you do not use the Tivoli Workload Scheduler for z/OS APPC server task you must log off and then log on to the system where the engine is active. This can be avoided by using the Tivoli Workload Scheduler for z/OS APPC server task.

17.5.2 Configure DVIPA for Tivoli Workload Scheduler for z/OS end-to-end server

To make sure that the engine can be moved from SC64 to either SC63 or SC65, Dynamic VIPA is used to define the IP address for the server task. This DVIPA IP address is defined in the profile data set pointed to by PROFILE DD-card in the TCPIP started task.

Example 17-9 on page 570 shows the VIPA definition that is used to define logical sysplex-wide IP addresses for the Tivoli Workload Scheduler for z/OS end-to-end server, engine JSC server.

Example 17-9 The VIPA definition

```
VIPADYNAMIC
  viparange define 255.255.255.248 9.12.6.104
ENDVIPADYNAMIC
PORT
   424    TCP TWSC      BIND 9.12.6.105
   5000   TCP TWSCJSC   BIND 9.12.6.106
   31282  TCP TWSCE2E   BIND 9.12.6.107
```

In this example, the first column under PORT is the port number, the third column is the name of the started task, and the fifth column is the logical sysplex-wide IP address. Port 424 is used for the Tivoli Workload Scheduler for z/OS tracker agent IP address, port 5000 for the Tivoli Workload Scheduler for z/OS JSC server task, and port 31282 is used for the Tivoli Workload Scheduler for z/OS end-to-end server task.

With these VIPA definitions, we have made a relationship between port number, started task name, and the logical IP address that can be used sysplex-wide.

The TWSCE2E host name and 31282 port number that is used for the Tivoli Workload Scheduler for z/OS end-to-end server is defined in the TOPLOGY HOSTNAME(TWSCE2E) initialization statement used by the TWSCE2E server and Tivoli Workload Scheduler for z/OS plan programs.

When the Tivoli Workload Scheduler for z/OS engine creates the Symphony file, the TWSCE2E host name and 31282 port number will be part of the Symphony file. The first-level domain manager (U100) and the backup domain manager (F101) will use this host name when they establish outbound IP connections to the Tivoli Workload Scheduler for z/OS server. The backup domain manager only establishes outbound IP connections to the Tivoli Workload Scheduler for z/OS server if it will take over the responsibilities for the first-level domain manager.

17.5.3 Configuring the backup domain manager for the first-level domain manager

In this section, we show how to configure a backup domain manager for a first-level domain manager. This scenario shows F100 FTA configured as the first-level domain manager and F101 FTA configured as the backup domain manager. The initial DOCREC definitions in Example 17-10 on page 571 show that F100 (in bold) is defined as the first-level domain manager.

Example 17-10 DOMREC definitions

```
/*********************************************************************/
/* DOMREC: Defines the domains in the distributed Tivoli Workload    */
/*         Scheduler network                                         */
/*********************************************************************/
/*-------------------------------------------------------------------*/
/* Specify one DOMREC for each domain in the distributed network.    */
/* With the exception of the master domain (whose name is MASTERDM   */
/* and consist of the TWS for z/OS engine).                          */
/*-------------------------------------------------------------------*/
DOMREC    DOMAIN(DM100)              /* Domain name for 1st domain    */
          DOMMNGR(F100)              /* Chatham FTA - domain manager  */
          DOMPARENT(MASTERDM)        /* Domain parent is MASTERDM     */
DOMREC    DOMAIN(DM200)              /* Domain name for 2nd domain    */
          DOMMNGR(F200)              /* Yarmouth FTA - domain manager */
          DOMPARENT(DM100)           /* Domain parent is DM100        */
```

The F101 fault-tolerant agent can be configured to be the backup domain manager simply by specifying the entries in bold in Example 17-11 in its CPUREC definition.

Example 17-11 Configuring F101 to be the backup domain manager

```
CPUREC    CPUNAME(F101)
          CPUTCPIP(31758)
          CPUUSER(tws)
          CPUDOMAIN(DM100)
          CPUSERVER(1)
          CPUFULLSTAT(ON)            /* Full status on for Backup DM  */
          CPURESDEP(ON)              /* Resolve dep. on for Backup DM */
```

With CPUFULLSTAT (full status information) and CPURESDEP (resolve dependency information) set to On, the Symphony file on F101 is updated with the same reporting and logging information as the Symphony file on F100. The backup domain manager will then be able to take over the responsibilities of the first-level domain manager.

Note: FixPack 04 introduces a new Fault-Tolerant Switch Feature, which is described in a PDF file named FaultTolerantSwitch.README.

The new Fault-Tolerant Switch Feature replaces and enhances the existing or traditional Fault-Tolerant Switch Manager for backup domain managers.

17.5.4 Switch to Tivoli Workload Scheduler backup domain manager

This scenario is divided into two parts:

- A short-term switch to the backup manager

  By “short-term switch,” we mean that we have switched back to the original domain manager before the current plan is extended or replanned.

- A long-term switch

  By a “long-term switch,” we mean that the switch to the backup manager will be effective across the current plan extension or replan.

Short-term switch to the backup manager

In this scenario, we issue a **`switchmgr`** command on the F101 backup domain manager. We verify that the F101 takes over the responsibilities of the old first-level domain manager.

The steps in the short-term switch scenario are:

1. Issue the switch command on the F101 backup domain manager.
2. Verify that the switch is done.

Step 1. Issue switch command on F101 backup domain manager

Before we perform the switch, we check the status of the workstations from a JSC instance pointing to the first-level domain manager (Figure 17-8).

Status of all Workstations

Name	Jobman Running	Link Status	Limit	Fence	Node	Node Port	CPU Type	Host	Run	Domain
OPCMASTER	No	LINKED	0	0	OTHER_MASTER	31281	MASTER		105	MASTERDM
F100	Yes		20	0	OTHER_MANAGER	31281	MANAGER	F100	105	DM100
F101	Yes	LINKED	20	0	OTHER_FTA	31281	FTA	F101	105	DM100
F102	Yes	LINKED	10	0	WNT_FTA	31281	FTA	F102	105	DM100
F200	Yes	LINKED	99	0	OTHER_MANAGER	31281	MANAGER	F200	105	DM200
F201	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F201	105	DM200
F202	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F202	105	DM200
F203	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F203	105	DM200

Figure 17-8 Status for workstations before the switch to F101

Note in Figure 17-8 that F100 is `MANAGER` (in the CPU Type column) for the DM100 domain. F101 is `FTA` (in the CPU Type column) in the DM100 domain.

To simulate that the F100 first-level domain manager is down or unavailable due to a system failure, we issue the switch manager command on the F101 backup domain manager. The switch manager command is initiated from the conman command line on F101:

```
conman switchmgr "DM100;F101"
```

In this example, DM100 is the domain and F101 is the fault-tolerant workstation we are going to switch to. The F101 fault-tolerant workstation responds with the message shown in Example 17-12.

Example 17-12 Messages showing switch has been initiated

```
switchmgr DM100;F101
AWS20710041I Service 2005 started on F101
AWS22020120  Switchmgr command executed from cpu F101 on cpu F101.
```

This indicates that the switch has been initiated.

It is also possible to initiate the switch from a JSC instance pointing to the F101 backup domain manager. Because we do not have a JSC instance pointing to the backup domain manager, we use the **conman switchmgr** command locally on the F101 backup domain manager.

For your information, we show how to initiate the switch from the JSC:

1. Double-click **Status of all Domains** in the Default Plan Lists in the domain manager JSC instance (TWSC-F100-Eastham) (Figure 17-9).

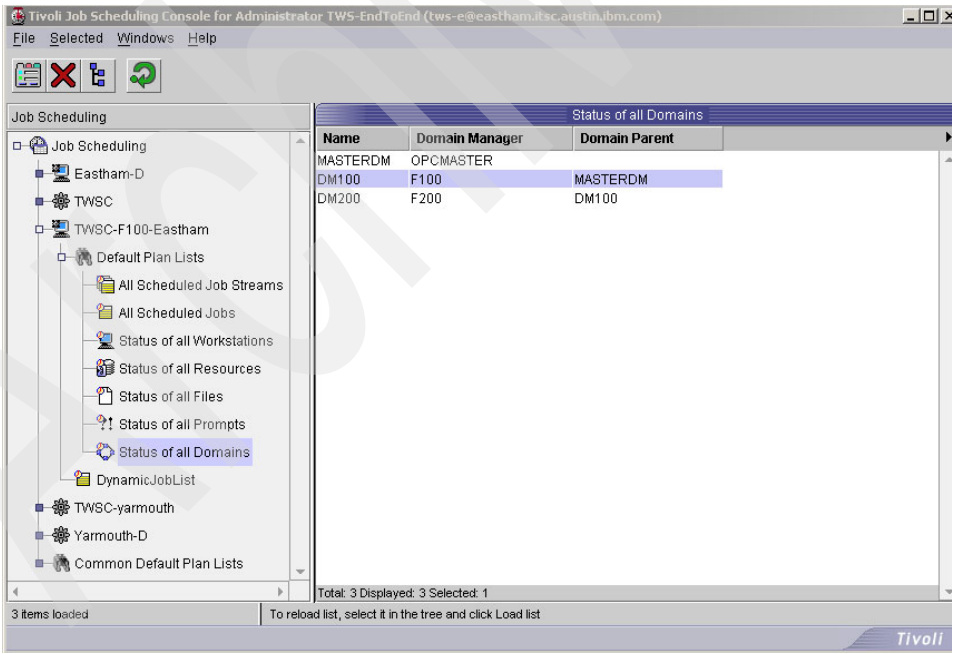


Figure 17-9 Status of all Domains list

2. Right-click the **DM100** domain to open the context menu shown in Figure 17-10, and select **Switch Manager**.

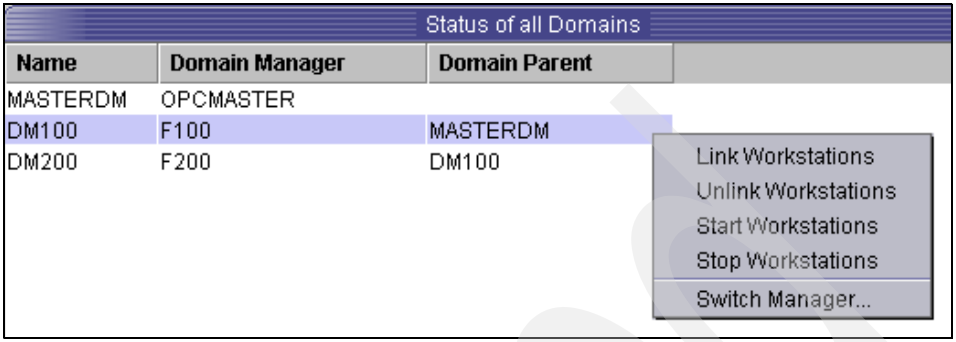


Figure 17-10 Context menu for the DM100 domain

3. The JSC shows a new pop-up window in which we can search for the agent we will switch to (Figure 17-11). Click the search button (the square box with three dots) to the right of the F100 domain.

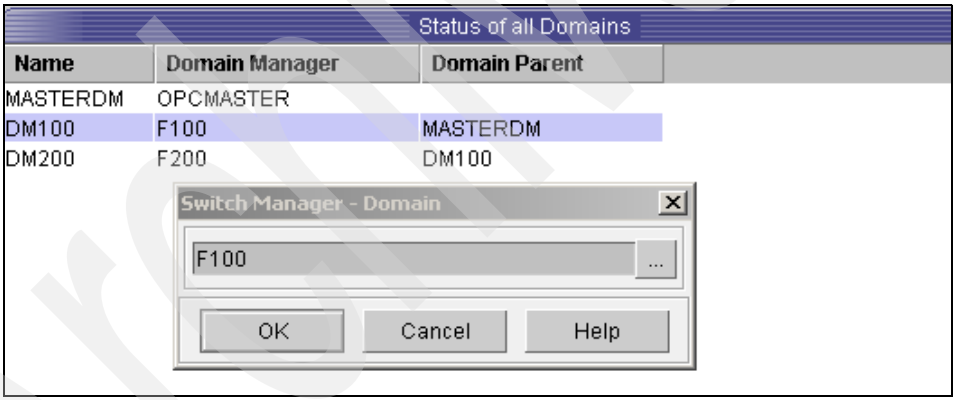


Figure 17-11 The Switch Manager: Domain search pop-up window

4. JSC opens the Find Workstation Instance pop-up window (Figure 17-12). Click **Start** .

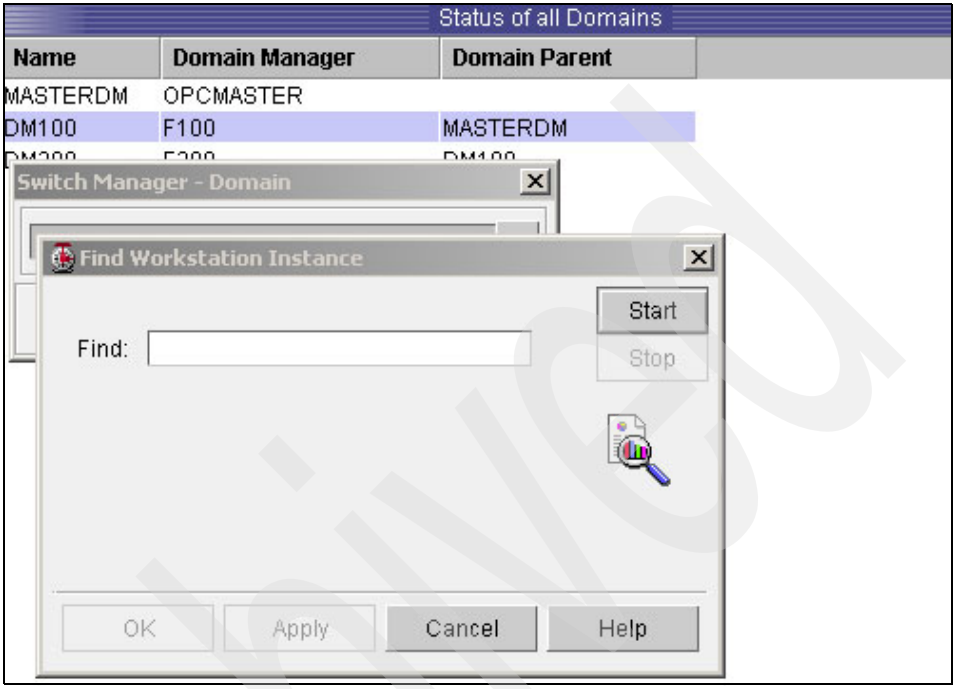


Figure 17-12 JSC Find Workstation Instance window

5. JSC opens the Find Workstation Instance pop-up window that lists all of the fault-tolerant workstations in the network (Figure 17-13). If we had specified a filter in the Find field shown in Figure 17-12 on page 575, this filter would be used to narrow the list of workstations that are shown.

   Mark the workstation to switch to F101 (in our example) and click **OK**.

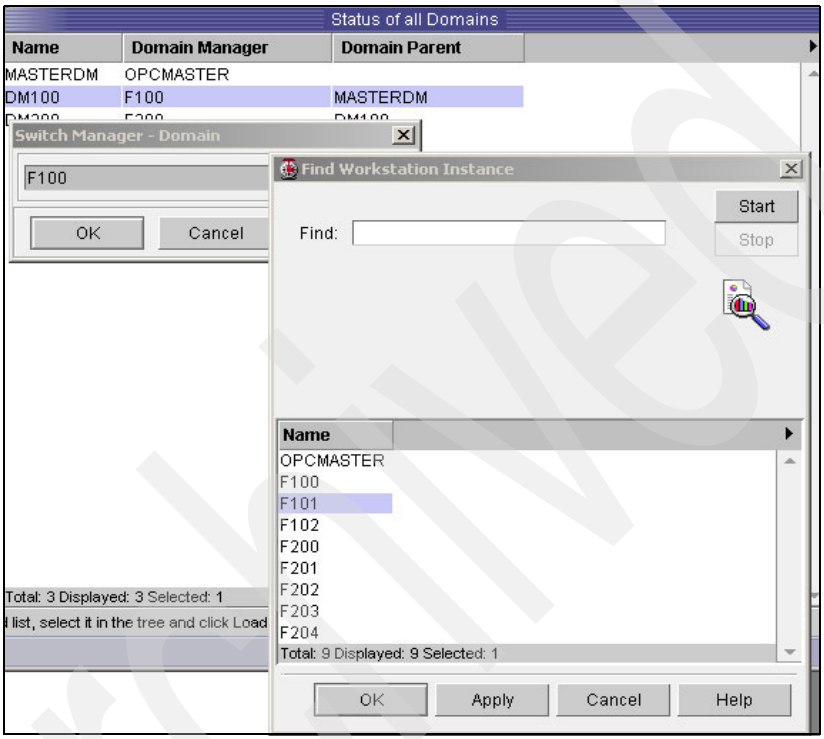


Figure 17-13 The result from Find Workstation Instance

6. Click **OK** in the Switch Manager - Domain pop-up window to initiate the switch. Note that the selected workstation (F101) appears in the pop-up window (Figure 17-14).

   The switch to F101 is initiated and Tivoli Workload Scheduler performs the switch.

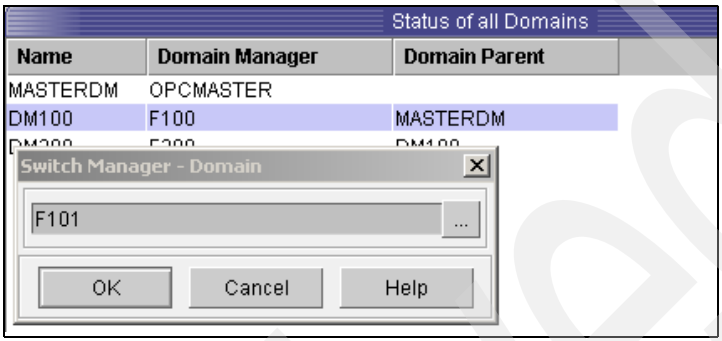


Figure 17-14 Switch Manager: Domain pop-up window with selected FTA

Note: With Tivoli Workload Scheduler for z/OS 8.2, you can now switch the domain manager using the WSSTAT TSO command on the mainframe. The Tivoli Workload Scheduler for z/OS guide *Managing the Workload,* SC32-1263, incorrectly states the syntax of this command. DOC APAR PQ93442 has been opened to correct the documentation.

If you prefer to work with the JSC, the previous method of switching will appeal to you. If you are a mainframe operator, you may prefer to perform this sort of task from the mainframe. The example below shows how to do switching using the WSSTAT TSO command instead.

In Example 17-13, the workstation F101 is instructed to become the new domain manager of the DM100 domain. The command is sent via the TWST tracker subsystem.

Example 17-13 Switching domain manager using the WSSTAT command

```
WSSTAT SUBSYS(TWST) WSNAME(F101) MANAGES(DM100)
```

If you prefer to work with the UNIX or Windows command line, Example 17-14 shows how to run the **switchmgr** command from conman.

Example 17-14 Switching domain manager using the switchmgr command

```
conman 'switchmgr DM100,F101'
```

Step 2. Verify that the switch is done

We check the status for the workstation using the JSC pointing to the old first-level domain manager, F100 (Figure 17-15).

Status of all Workstations

Name	Jobman Running	Link Status	Limit	Fence	Node	Node Port	CPU Type	Host	Run	Domain
OPCMASTER	No	UNLINKED	0	0	OTHER_MASTER	31281	MASTER		112	MASTER
F100	Yes		20	0	OTHER_FTA	31281	FTA	F100	112	DM100
F101	Yes	LINKED	20	0	OTHER_MANAGER	31281	MANAGER	F101	112	DM100
F102	Yes	LINKED	10	0	WNT_FTA	31281	FTA	F102	112	DM100
F200	Yes	LINKED	99	0	OTHER_MANAGER	31281	MANAGER	F200	112	DM200
F201	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F201	112	DM200
F202	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F202	112	DM200
F203	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F203	112	DM200

Figure 17-15 Status for the workstations after the switch to F101

In Figure 17-15, it can be verified that F101 is now `MANAGER` (under CPU Type) for the DM100 domain (the Domain column). The F100 is changed to an `FTA` (shown in the CPU Type column).

The OPCMASTER workstation has the status unlinked (as shown in the Link Status column in Figure 17-15). This status is correct, as we are using the JSC instance pointing to the F100 workstation. The OPCMASTER has a linked status on F101, as expected.

Switching to the backup domain manager takes some time (so be patient) because the switch manager command stops the backup domain manager and restarts it as the domain manager. All domain member fault-tolerant workstations are informed about the switch, and the old domain manager is converted to a fault-tolerant agent in the domain. The fault-tolerant workstations use the switch information to update their Symphony file with the name of the new domain manager, then they stop and restart to link to the new domain manager.

On rare occasions, the link status is not shown correctly in the JSC after a switch to the backup domain manager. If this happens, try to link the workstation manually by right-clicking the workstation and clicking **Link** in the pop-up window.

Note: To reactivate F100 as the domain manager, simply perform a switch manager to F100 or run Symphony redistribute. The F100 will also be reinstated as the domain manager when you run the extend or replan programs.

Long-term switch to the backup manager

The identification of domain managers is placed in the Symphony file. If a switch domain manager command is issued, the old domain manager name will be replaced with the new (backup) domain manager name in the Symphony file.

For the switch to the backup domain manager to be effective across Tivoli Workload Scheduler for z/OS plan extension or replan, we have to update the DOMREC definition. This is also the case if we redistribute the Symphony file from Tivoli Workload Scheduler for z/OS.

The plan program reads the DOMREC definitions and creates a Symphony file with domain managers and fault-tolerant agents accordingly. If the DOMREC definitions are not updated to reflect the switch to the backup domain manager, the old domain manager will automatically resume domain management possibilities.

The steps in the long-term switch scenario are:

1. Issue the switch command on the F101 backup domain manager.
2. Verify that the switch is done.
3. Update the DOMREC definitions used by the TWSCE2E server and the Tivoli Workload Scheduler for z/OS plan programs.
4. Run the replan plan program in Tivoli Workload Scheduler for z/OS.
5. Verify that the switched F101 is still the domain manager.

Step 1. Issue switch command on F101 backup domain manager

The switch command is done as described in “Step 1. Issue switch command on F101 backup domain manager” on page 572.

Step 2. Verify that the switch is done

We check the status of the workstation using the JSC pointing to the old first-level domain manager, F100 (Figure 17-16).

Status of all Workstations

Name	Jobman Running	Link Status	Limit	Fence	Node	Node Port	CPU Type	Host	Run	Domain
OPCMASTER	No	UNLINKED	0	0	OTHER_MASTER	31281	MASTER		112	MASTER
F100	Yes		20	0	OTHER_FTA	31281	FTA	F100	112	DM100
F101	Yes	LINKED	20	0	OTHER_MANAGER	31281	MANAGER	F101	112	DM100
F102	Yes	LINKED	10	0	WNT_FTA	31281	FTA	F102	112	DM100
F200	Yes	LINKED	99	0	OTHER_MANAGER	31281	MANAGER	F200	112	DM200
F201	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F201	112	DM200
F202	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F202	112	DM200
F203	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F203	112	DM200

Figure 17-16 Status of the workstations after the switch to F101

From Figure 17-16 it can be verified that F101 is now `MANAGER` (see CPU Type column) for the DM100 domain (see the Domain column). F100 is changed to an `FTA` (see the CPU Type column).

The OPCMASTER workstation has the status unlinked (see the Link Status column in Figure 17-16). This status is correct, as we are using the JSC instance

pointing to the F100 workstation. The OPCMASTER has a linked status on F101, as expected.

Step 3. Update the DOMREC definitions for server and plan program

We update the DOMREC definitions, so F101 will be the new first-level domain manager (Example 17-15).

Example 17-15 DOMREC definitions

```
/*********************************************************************/
/* DOMREC: Defines the domains in the distributed Tivoli Workload    */
/*         Scheduler network                                         */
/*********************************************************************/
/*-------------------------------------------------------------------*/
/* Specify one DOMREC for each domain in the distributed network.    */
/* With the exception of the master domain (whose name is MASTERDM   */
/* and consist of the TWS for z/OS engine).                      */
/*-------------------------------------------------------------------*/
DOMREC    DOMAIN(DM100)              /* Domain name for 1st domain    */
          DOMMNGR(F101)              /* Chatham FTA - domain manager  */
          DOMPARENT(MASTERDM)        /* Domain parent is MASTERDM    */
DOMREC    DOMAIN(DM200)              /* Domain name for 2nd domain    */
          DOMMNGR(F200)              /* Yarmouth FTA - domain manager */
          DOMPARENT(DM100)           /* Domain parent is DM100       */
```

The DOMREC DOMNGR(F101) defines the name of the first-level domain manager. This is the only change needed in the DOMREC definition.

We did create an extra member in the EQQPARM data set and called it TPSWITCH. This member has the updated DOMREC definitions to be used when we have a long-term switch. In the EQQPARM data set, we have three members: TPSWITCH (F101 is domain manager), TPNORM (F100 is domain manager), and TPDOMAIN (the member used by TWSCE2E and the plan programs).

Before the plan programs are executed, we replace the TPDOMAIN member with the TPSWITCH member. When F100 is to be the domain manager again we simply replace the TPDOMAIN member with the TPNORM member.

Tip: If you let your system automation (for example, System Automation/390) handle the switch to the backup domain manager, you can automate the entire process.

- System automation replaces the EQQPARM members.
- System automation initiates the switch manager command remotely on the fault-tolerant workstation.
- System automation resets the definitions when the original domain manager is ready be activated.

Step 4. Run replan plan program in Tivoli Workload Scheduler for z/OS

We submit a replan plan program (job) using option 3.1 from ISPF in the Tivoli Workload Scheduler for z/OS engine and verify the output. Example 17-16 shows the messages in EQQMLOG.

Example 17-16 EQQMLOG

```
EQQZ014I MAXIMUM RETURN CODE FOR PARAMETER MEMBER TPDOMAIN IS: 0000
EQQ3005I CPU F101 IS SET AS DOMAIN MANAGER OF FIRST LEVEL
EQQ3030I DOMAIN MANAGER F101 MUST HAVE SERVER ATTRIBUTE SET TO BLANK
EQQ3011I CPU F200 SET AS DOMAIN MANAGER
EQQZ013I NOW PROCESSING PARAMETER LIBRARY MEMBER TPUSER
EQQZ014I MAXIMUM RETURN CODE FOR PARAMETER MEMBER TPUSER   IS: 0000
```

The F101 fault-tolerant workstation is the first-level domain manager.

The EQQ3030I message is due to the CPUSERVER(1) specification in the CPUREC definition for the F101 workstation. The CPUSERVER(1) specification is used when F101 is running as a fault-tolerant workstation managed by the F100 domain manager.

Step 5. Verify that the switched F101 is still domain manager

Finally, we verify that F101 is the domain manager after the replan program has finished and the Symphony file is distributed (Figure 17-17).

Status of all Workstations

Name	Jobman Running	Link Status	Limit	Fence	Node	Node Port	CPU Type	Host	Run	Domain
OPCMASTER	No	UNLINKED	0	0	OTHER_MASTER	31281	MASTER		114	MASTERD
F100	Yes		20	0	OTHER_FTA	31281	FTA	F100	114	DM100
F101	Yes	LINKED	20	0	OTHER_MANAGER	31281	MANAGER	F101	114	DM100
F102	Yes	LINKED	10	0	WNT_FTA	31281	FTA	F102	114	DM100
F200	Yes	LINKED	99	0	OTHER_MANAGER	31281	MANAGER	F200	114	DM200
F201	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F201	114	DM200
F202	Yes	LINKED	20	0	WNT_FTA	31281	FTA	F202	114	DM200

Figure 17-17 Workstations status after Tivoli Workload Scheduler for z/OS replan

From Figure 17-17 on page 581, it can be verified that F101 is still MANAGER (in the CPU Type column) for the DM100 domain (in the Domain column). The CPU type for F100 is FTA.

The OPCMASTER workstation has the status unlinked (the Link Status column). This status is correct, as we are using the JSC instance pointing to the F100 workstation. The OPCMASTER has a linked status on F101, as expected.

Note: To reactivate F100 as a domain manager, simply do a switch manager to F100 or Symphony redistribute. The F100 will also be reinstated as domain manager when you run the extend or replan programs. Remember to change the DOMREC definitions before the plan programs are executed; otherwise, the Symphony file will be redistributed.

17.5.5 Implementing Tivoli Workload Scheduler high availability on high-availability environments

You can also use high-availability environments such as High Availability Cluster Multi-Processing (HACMP) or Microsoft Cluster (MSCS) to implement fail-safe Tivoli Workload Scheduler workstations.

The redbook *High Availability Scenarios with IBM Tivoli Workload Scheduler and IBM Tivoli Framework*, SG24-6632, discusses these scenarios in detail, so we refer you to it for implementing Tivoli Workload Scheduler high availability using HACMP or MSCS.

17.6 Backup and maintenance guidelines for FTAs

In this section, we discuss some important backup and maintenance guidelines for Tivoli Workload Scheduler fault-tolerant agents (workstations) in an end-to-end scheduling environment.

17.6.1 Backup of the Tivoli Workload Scheduler FTAs

To make sure that you can recover from disk or system failures on the system where the Tivoli Workload Scheduler engine is installed, you should make a daily or weekly backup of the installed engine.

The backup can be done in several ways. You probably already have some backup policies and routines implemented for the system where the Tivoli Workload Scheduler engine is installed. These backups should be extended to make a backup of files in the <TWShome> and the <TWShome/..> directories.

We suggest that you have a backup of all of the Tivoli Workload Scheduler files in the <TWShome> and <TWShome/..> directories. If the Tivoli Workload Scheduler engine is running as a fault-tolerant workstation in an end-to-end network, it should be sufficient to make the backup on a weekly basis.

When deciding how often a backup should be generated, consider:

- Are you using parameters on the Tivoli Workload Scheduler agent?

  If you are using parameters locally on the Tivoli Workload Scheduler agent and do not have a central repository for the parameters, you should consider making daily backups.

- Are you using specific security definitions on the Tivoli Workload Scheduler agent?

  If you are using specific security file definitions locally on the Tivoli Workload Scheduler agent and do not have a central repository for the security file definitions, you should consider making daily backups.

Another approach is to make a backup of the Tivoli Workload Scheduler agent files, at least before making any changes to the files. For example, the changes can be updates to configuration parameters or a patch update of the Tivoli Workload Scheduler agent.

17.6.2 Stdlist files on Tivoli Workload Scheduler FTAs

Tivoli Workload Scheduler fault-tolerant agents save job logs on the system where the jobs run. These job logs are stored in a directory named <twshome>/stdlist. In the stdlist (standard list) directory, there will be subdirectories with the name *ccyy.mm.dd* (where cc is the century, yy is the year, mm is the month, and dd is the date).

This subdirectory is created daily by the Tivoli Workload Scheduler netman process when a new Symphony file (Sinfonia) is received on the fault-tolerant agent. The Symphony file is generated by the Tivoli Workload Scheduler for z/OS Controller plan program in the end-to-end scheduling environment.

The *ccyy.mm.dd* subdirectory contains a job log for each job that is executed on a particular production day, as seen in Example 17-17.

Example 17-17 Files in a stdlist/ccyy.mm.dd directory

```
019502.0908     File with job log for job with process no. 19502 run at 09.08
019538.1052     File with job log for job with process no. 19538 run at 10.52
038380.1201     File with job log for job with process no. 38380 run at 12.01
```

These log files are created by the Tivoli Workload Scheduler job manager process (jobman) and will remain there until deleted by the system administrator.

Tivoli Workload Scheduler also logs messages from its own programs. These messages are stored in a subdirectory of the stdlist directory called logs.

The easiest way to maintain the growth of these directories is to decide how long the log files are needed and schedule a job under Tivoli Workload Scheduler for z/OS control, which removes any file older than the given number of days. The Tivoli Workload Scheduler **rmstdlist** command can perform the deletion of stdlist files and remove or display files in the stdlist directory based on age of the files:

```
rmstdlist [-v |-u]
rmstdlist [-p] [age]
```

In these commands, the arguments are:

-v Displays the command version and exits.

-u Displays the command usage information and exits.

-p Displays the names qualifying standard list file directories. No directory or files are removed. If you do not specify -p, the qualifying standard list files are removed.

age The minimum age, in days, for standard list file directories to be displayed or removed. The default is 10 days.

We suggest that you run the **rmstdlist** command daily on all of your FTAs. This command can be defined in a job in a job stream and scheduled by Tivoli Workload Scheduler for z/OS. You may need to save a backup copy of the stdlist files, for example, for internal revision or due to company policies. If this is the case, a backup job can be scheduled to run just before the rmstdlist job.

17.6.3 Auditing log files on Tivoli Workload Scheduler FTAs

The auditing function can be used to track changes to the Tivoli Workload Scheduler plan (the Symphony file) on FTAs. Plan auditing is enabled by the TOPLOGY PLANAUDITLEVEL parameter, described below.

```
PLANAUDITLEVEL(0|1)
```

This parameter enables or disables plan auditing for distributed agents. Valid values are 0 to disable plan auditing and 1 to activate plan auditing. Auditing information is logged to a flat file in the TWShome/audit/plan directory. Each Tivoli Workload Scheduler workstation maintains its own log. Only actions are logged in the auditing file, not the success or failure of any action. If you change the value, you also need to restart the Tivoli Workload Scheduler for z/OS server and renew the Symphony file.

After plan auditing has been enabled, modification to the Tivoli Workload Scheduler plan (the Symphony) on an FTA will be added to the plan directory on that workstation (<*date*> is in ccyymmdd format):

```
<TWShome>/audit/plan/<date>
```

We suggest that you clean out the audit database and plan directories regularly, daily if necessary. The cleanout in the directories can be defined in a job in a job stream and scheduled by Tivoli Workload Scheduler for z/OS. You may need to save a backup copy of the audit files (for internal revision or due to company policies, for example). If so, a backup job can be scheduled to run just before the cleanup job.

17.6.4 Monitoring file systems on Tivoli Workload Scheduler FTAs

It is easier to deal with file system problems before they happen. If your file system fills up, Tivoli Workload Scheduler will no longer function and your job processing will stop. To avoid problems, monitor the file systems containing your Tivoli Workload Scheduler home directory and /tmp. For example, if you have a 2 GB file system, you might want a warning at 80%, but if you have a smaller file system, you will need a warning when a lower percentage fills up. We cannot give you an exact percentage at which to be warned. This depends on many variables that change from installation to installation (or company to company).

Monitoring or testing for the percentage of the file system can be done by, for example, IBM Tivoli Monitoring and IBM Tivoli Enterprise Console® (TEC).

Example 17-18 shows a sample shell script that tests for the percentage of the Tivoli Workload Scheduler file system filled and report back if it is over 80%.

Example 17-18 Monitoring script

```
/usr/bin/df -P /dev/lv01  | grep TWS >> tmp1$$
/usr/bin/awk '{print $5}' tmp1$$ > tmp2$$
/usr/bin/sed 's/%$//g' tmp2$$ > tmp3$$
x=`cat tmp3$$`
i=`expr $x \> 80`
echo "This file system is less than 80% full." >> tmp4$$
if [ "$i" -eq 1 ]; then
       echo "This file system is over 80% full. You need to
             remove schedule logs and audit logs from the
             subdirectories in the file system." > tmp4$$
       fi
cat tmp4$$
rm tmp1$$ tmp2$$ tmp3$$ tmp4$$
```

17.6.5 Central repositories for important Tivoli Workload Scheduler files

Several files are important for use of Tivoli Workload Scheduler and for the daily Tivoli Workload Scheduler production workload if you are running a Tivoli Workload Scheduler master domain manager or a Tivoli Workload Scheduler for z/OS end-to-end server. Managing these files across several Tivoli Workload Scheduler workstations can be a cumbersome and time-consuming task. Using central repositories for these files can save time and make your management more effective.

Script files

Scripts (or the JCL) are very important objects when doing job scheduling on the Tivoli Workload Scheduler fault-tolerant agents. It is the scripts that actually perform the work or the job on the agent system, such as updating the payroll database or the customer inventory database.

The job definition for distributed jobs in Tivoli Workload Scheduler or Tivoli Workload Scheduler for z/OS contains a pointer (the path or directory) to the script. The script by itself is placed locally on the fault-tolerant agent. Because the fault-tolerant agents have a local copy of the plan (Symphony) and the script to run, they can continue running jobs on the system even if the connection to the Tivoli Workload Scheduler master or the Tivoli Workload Scheduler for z/OS Controller is broken. This way we have fault tolerance on the workstations.

Managing scripts on several Tivoli Workload Scheduler fault-tolerant agents and making sure that you always have the correct versions on every fault-tolerant agent can be a time-consuming task. You also must ensure that the scripts are protected so that they cannot be updated by the wrong person. Pure protected scripts can cause problems in your production environment if someone has changed something without notifying the responsible planner or change manager.

We suggest placing all scripts that are used for production workload in one common script repository. The repository can be designed in different ways. One way could be to have a subdirectory for each fault-tolerant workstation (with the same name as the name on the Tivoli Workload Scheduler workstation).

All changes to scripts are made in this production repository. On a daily basis, for example, just before the plan is extended, the master scripts in the central repository are distributed to the fault-tolerant agents. The daily distribution can be handled by a Tivoli Workload Scheduler scheduled job. This job can be defined as predecessor to the plan extend job.

This approach can be made even more advanced by using a software distribution application to handle the distribution of the scripts. The software distribution application can help keep track of different versions of the same script. If you

encounter a problem with a changed script in a production shift, you can simply ask the software distribution application to redistribute a previous version of the same script and then rerun the job.

Security files

The Tivoli Workload Scheduler security file, discussed in detail 17.7, “Security on fault-tolerant agents” on page 587, is used to protect access to Tivoli Workload Scheduler database and plan objects. On every Tivoli Workload Scheduler engine (such as domain manager and fault-tolerant agent), you can issue conman commands for the plan and composer commands for the database. Tivoli Workload Scheduler security files are used to ensure that the right people have the right access to objects in Tivoli Workload Scheduler.

Security files can be created or modified on every local Tivoli Workload Scheduler workstation, and they can be different from workstation to workstation.

We suggest having a common security strategy for all Tivoli Workload Scheduler workstations in your Tivoli Workload Scheduler network (and end-to-end network). This way, the security file can be placed centrally and changes are made only in the central security file. If the security file has been changed, it is simply distributed to all Tivoli Workload Scheduler workstations in your Tivoli Workload Scheduler network.

17.7 Security on fault-tolerant agents

In this section, we offer an overview of how security is implemented on Tivoli Workload Scheduler fault-tolerant agents (including domain managers). For more details, see the *IBM Tivoli Workload Scheduler Planning and Installation Guide,* SC32-1273.

Figure 17-18 on page 588 shows the security model on Tivoli Workload Scheduler fault-tolerant agents. When a user attempts to display a list of defined jobs, submit a new job stream, add a new resource, or any other operation related to the Tivoli Workload Scheduler plan or databases, Tivoli Workload Scheduler performs a check to verify that the user is authorized to perform that action.

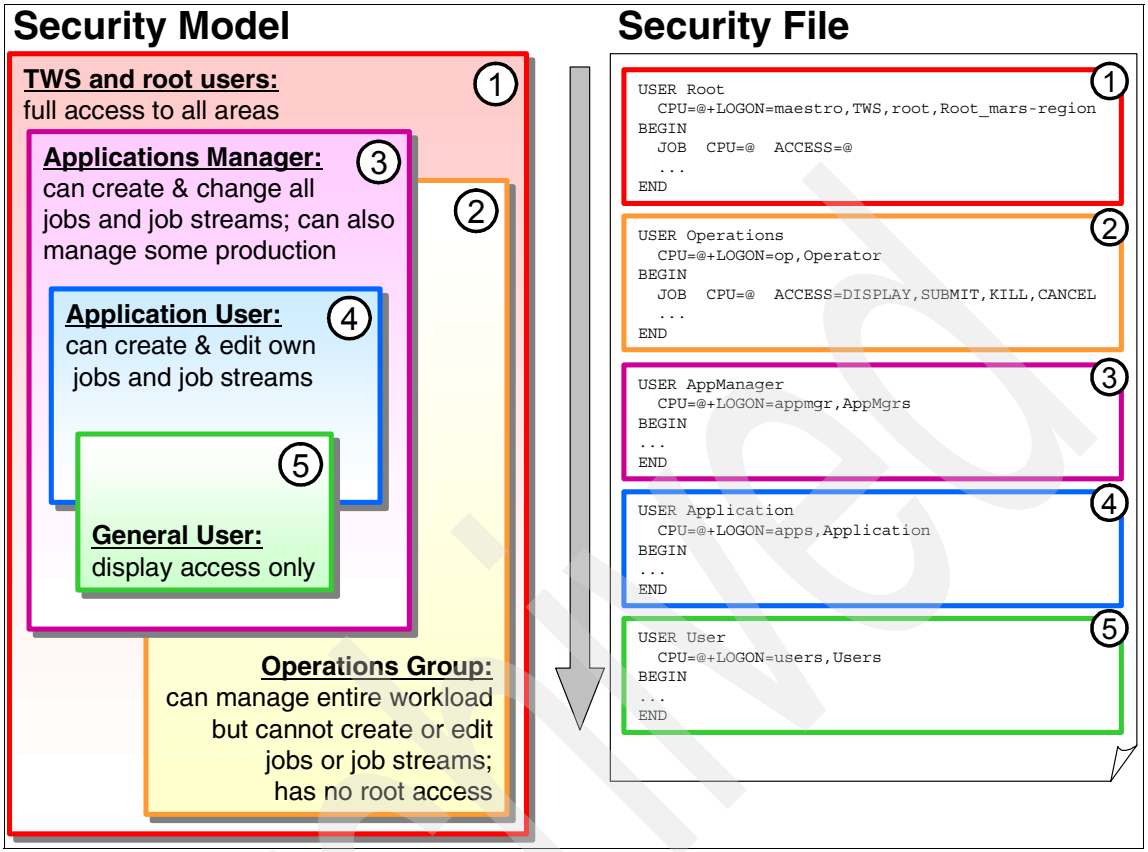


Figure 17-18 Sample security setup

Tivoli Workload Scheduler users have different roles in the organization, so the Tivoli Workload Scheduler security model you implement should reflect these roles. You can think of the different groups of users as nested boxes, as in Figure 17-18. The largest box represents the highest access, granted to only the Tivoli Workload Scheduler user and the root user. The smaller boxes represent more restricted roles, with correspondingly restricted access. Each group that is represented by a box in the figure would have a corresponding stanza in the security file. Tivoli Workload Scheduler programs and commands read the security file to determine whether the user has the access that is required to perform an action.

17.7.1 The security file

Each workstation in a Tivoli Workload Scheduler network has its own security file. These files can be maintained independently on each workstation, or you

can keep a single centralized security file on the master and copy it periodically to the other workstations in the network.

At installation time, a default security file is created that allows unrestricted access to only the Tivoli Workload Scheduler user (and, on UNIX workstations, the root user). If the security file is accidentally deleted, the root user can generate a new one.

If you have one security file for a network of agents, you may wish to make a distinction between the root user on a fault-tolerant agent and the root user on the master domain manager. For example, you can restrict local users to performing operations that affect only the local workstation, while permitting the master root user to perform operations that affect any workstation in the network.

A template file named TWShome/config/Security is provided with the software. During installation, a copy of the template is installed as TWShome/Security, and a compiled copy is installed as TWShome/../unison/Security.

Security file stanzas

The security file is divided into one or more *stanzas*. Each stanza limits access at three different levels:

- *User attributes* appear between the USER and BEGIN statements and determine whether a stanza applies to the user attempting to perform an action.
- *Object attributes* are listed, one object per line, between the BEGIN and END statements. Object attributes determine whether an object line in the stanza matches the object the user is attempting to access.
- *Access rights* appear to the right of each object listed, after the ACCESS statement. Access rights are the specific actions that the user is allowed to take on the object.

Important: Because only a subset of conman commands is available on FTAs in an end-to-end environment, some of the ACCESS rights that would be applicable in an ordinary non-end-to-end Tivoli Workload Scheduler network will not be applicable in an end-to-end network.

The steps of a security check

The steps of a security check reflect the three levels listed above:

1. Identify the user who is attempting to perform an action.
2. Determine the type of object being accessed.
3. Determine whether the requested access should be granted to that object.

Step 1: Identify the user

When a user attempts to perform any Tivoli Workload Scheduler action, the security file is searched from top to bottom to find a stanza whose user attributes match the user attempting to perform the action. If no match is found in the first stanza, the user attributes of the next stanza are searched. If a stanza is found whose user attributes match that user, that stanza is selected for the next part of the security check. If no stanza in the security file has user attributes that match the user, access is denied.

Step 2: Determine the type of object being accessed

After the user has been identified, the stanza that applies to that user is searched, top-down, for an object attribute that matches the type of object the user is trying to access. Only that particular stanza (between the BEGIN and END statements) is searched for a matching object attribute. If no matching object attribute is found, access is denied.

Step 3: Determine whether access is granted to that object

If an object attribute is located that corresponds to the object that the user is attempting to access, the access rights following the ACCESS statement on that line in the file are searched for the action that the user is attempting to perform. If this access right is found, then access is granted. If the access right is not found on this line, then the rest of the stanza is searched for other object attributes (other lines) of the same type, and this step is repeated for each of these.

Figure 17-19 on page 591 illustrates the steps of the security check algorithm.

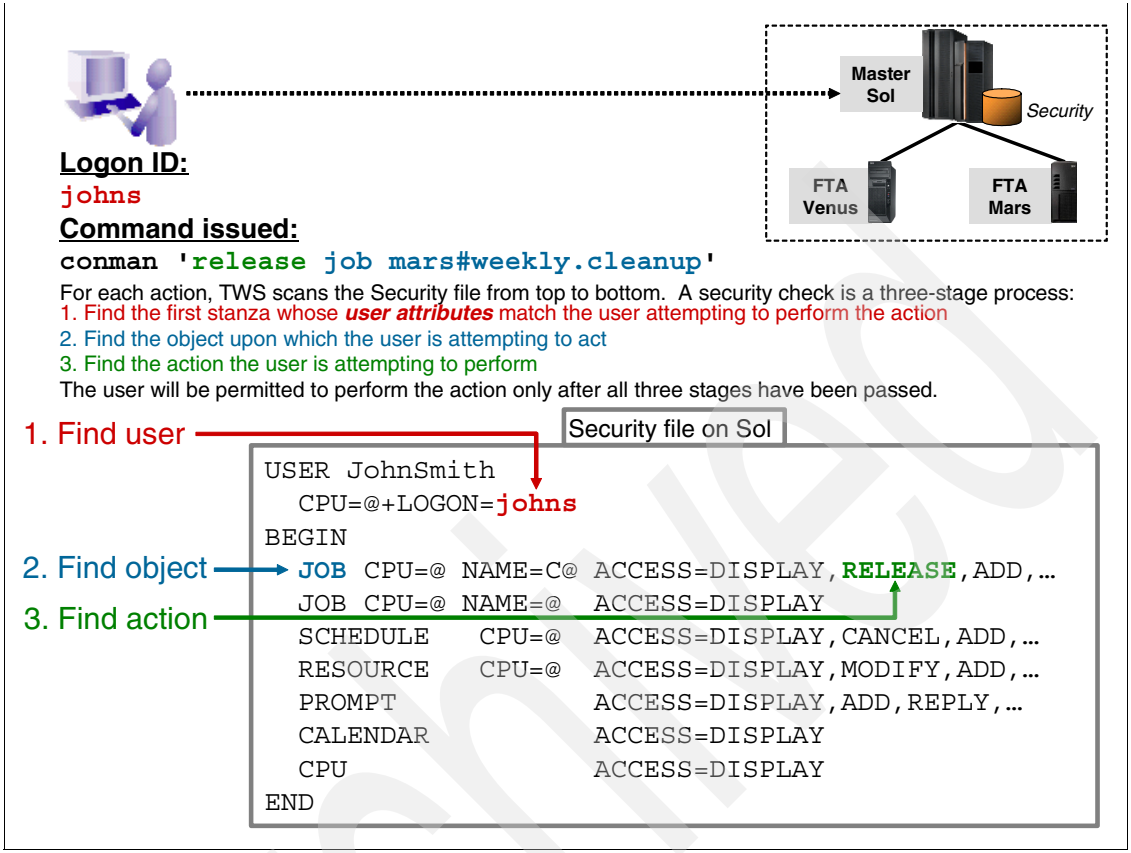


Figure 17-19 Example of a Tivoli Workload Scheduler security check

17.7.2 Sample security file

Here are some things to note about the security file stanza (Example 17-19 on page 592):

- `mastersm` is an arbitrarily chosen name for this group of users.
- The example security stanza above would match a user that logs on to the master (or to the Framework via JSC), where the user name (or TMF Administrator name) is `maestro`, `root`, or `Root_london-region`.
- These users have full access to jobs, jobs streams, resources, prompts, files, calendars, and workstations.
- The users have full access to all parameters except those whose names begin with r (`parameter name=@ ~ name=r@ access=@`).

For NT user definitions (userobj), the users have full access to objects on all workstations in the network.

Example 17-19 Sample security file

```
############################################################
#Sample Security File
############################################################
#(1)APPLIES TO MAESTRO OR ROOT USERS LOGGED IN ON THE
#MASTER DOMAIN MANAGER OR FRAMEWORK.
user mastersm cpu=$master,$framework
+logon=maestro,root,Root_london-region
begin
#OBJECT ATTRIBUTES ACCESS CAPABILITIES
#-----------------------------------------
job                         access=@
schedule                    access=@
resource                    access=@
prompt                      access=@
file                        access=@
calendar                    access=@
cpu                         access=@
parameter name=@ ~ name=r@ access=@
userobj   cpu=@ + logon=@ access=@
end
```

Creating the security file

To create user definitions, edit the template file TWShome/Security. Do not modify the original template in TWShome/config/Security. Then use the **makesec** command to compile and install a new operational security file. After it is installed, you can make further modifications by creating an editable copy of the operational file with the **dumpsec** command.

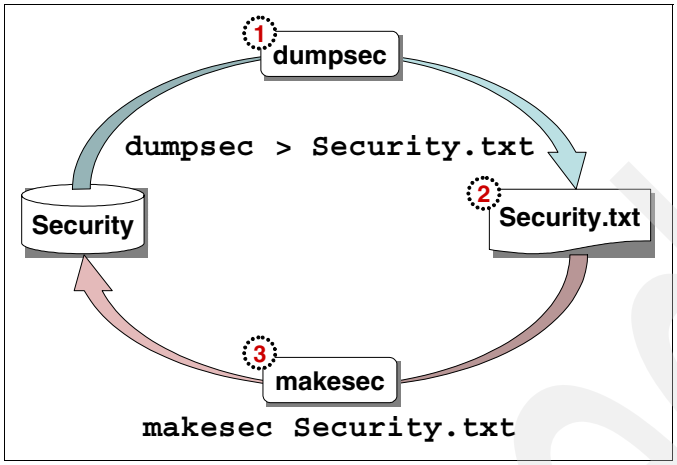


Figure 17-20 The dumpsec and makesec commands

The dumpsec command

The **dumpsec** command takes the existing Security file, and exports a text version to the standard output. The user must have display access to the security file.

- Synopsis:

```
dumpsec -v | -u
dumpsec > security-file
```

- Description:

  If no arguments are specified, the operational security file (Security) is dumped. To create an editable copy of a security file, redirect the output of the command to another file, as shown in the example above.

- Arguments
  - `-v` displays command version information only.
  - `-u` displays command usage information only.
  - `security-file` specifies the name of the text file to write.

The makesec command

The **makesec** command essentially does the opposite of what the **dumpsec** command does. The **makesec** command takes a text security file, checks its syntax, compiles it into a binary security file, and installs the new binary file as the active security file. Changes to the security file take effect when Tivoli Workload Scheduler is stopped and restarted. Affected programs are:

- conman
- Composer
- Tivoli Workload Scheduler connectors

Simply exit the programs. The next time they are run, the new security definitions will be recognized. Tivoli Workload Scheduler connectors must be stopped using the **wmaeutil** command before changes to the security file will take effect for users of JSC. The connectors will automatically restart as needed.

The user must have modify access to the security file.

Note: On Windows NT, the connector processes must be stopped (using the **wmaeutil** command) before the **makesec** command will work correctly.

- Synopsis:

```
makesec -v | -u
makesec [-verify] in-file
```

- Description:

  The **makesec** command compiles the specified file and makes it the active security file (Security). If the -verify argument is specified, the file is checked for correct syntax, but it is not activated.

- Arguments:
  - `-v` displays command version information only.
  - `-u` displays command usage information only.
  - `-verify` checks the syntax of the user definitions in the in-file only. The file is not installed as the security file. (Syntax checking is performed automatically when the security file is installed.)
  - `in-file` specifies the name of a file or set of files containing user definitions. A file name expansion pattern is permitted.

Example of dumpsec and makesec

Example 17-20 creates an editable copy of the active security file in a file named Security.conf, modifies the user definitions with a text editor, then compiles Security.conf and replaces the active security file.

Example 17-20 Using dumpsec and makesec

```
dumpsec > Security.conf
vi Security.conf
(Here you would make any required modifications to the Security.conf file)
makesec Security.conf
```

Note: Add the Tivoli Administrator to the Tivoli Workload Scheduler security file after you have installed the Tivoli Management Framework and Tivoli Workload Scheduler connector.

Configuring Tivoli Workload Scheduler security for the Tivoli Administrator

In order to use the Job Scheduling Console on a master or on an FTA, the Tivoli Administrator user (or users) must be defined in the security file of that master or FTA. The $framework variable can be used as a user attribute in place of a specific workstation. This indicates a user logging in via the Job Scheduling Console.

17.8 End-to-end scheduling tips and tricks

In this section, we provide some tips, tricks, and troubleshooting suggestions for the end-to-end scheduling environment.

17.8.1 File dependencies in the end-to-end environment

Use the filewatch.sh program that is delivered with Tivoli Workload Scheduler. Description of usage and parameters is first in the filewatch.sh program.

In an ordinary (non-end-to-end) Tivoli Workload Scheduler network—one in which the MDM is a UNIX or Windows workstation—it is possible to create a file dependency on a job or job stream; this is not possible in an end-to-end network because the controlling system is Tivoli Workload Scheduler for z/OS.

It is very common to use files as triggers or *predecessors* to job flows on non-mainframe systems such as UNIX servers.

Tivoli Workload Scheduler 8.2 includes TWSHOME/bin/filewatch.sh, a sample script that can be used to check for the existence of files. You can configure the script to check periodically for the file, just as with a real Tivoli Workload Scheduler file dependency. By defining a job that runs filewatch.sh, you can implement a file dependency.

Here is an overview of the filewatch.sh script:

- Synopsis:

```
filewatch.sh -kd test_options
             -fl path_name
             -dl deadline | -nd
             -int interval
             [ -rc return_code ]
             [ -tsk task ]
```

- Description:

  filewatch.sh checks periodically for the existence of a file.

- Arguments:
  - `-kd` The options to pass to the test command. See the man page for the test command for a list of allowed values.
  - `-fl` Path name of the file (or directory) to look for.
  - `-dl` The deadline period in seconds; cannot be used together with -nd.
  - `-nd` No deadline; cannot be used together with -dl.
  - `-int` The interval between file checks.
  - `-rc` The return code to exit with if the file is not found by the deadline.
  - `-tsk` The task to run if the file is found.

- Examples:
  - In this example, the script checks for file /tmp/filew01 every 15 seconds indefinitely:

    ```
    JOBSCR('/tws/bin/filewatch.sh -kd f -fl /tmp/filew01 -int 15
    -nd')
    ```

  - In this example, the script checks for file /tmp/filew02 every 15 seconds for 60 seconds. If file is not there 60 seconds after the check has started, the script will end with return code 12:

    ```
    JOBSCR('/tws/bin/filewatch.sh -kd f  -fl /tmp/filew02 -int 15
    -dl 60 -rc 12')
    ```

Figure 17-21 on page 597 shows how the filewatch script might be used as a predecessor to the job that will process the file being “watched.” This way you can make sure that the file to be processed is there before running the job that will process the file.

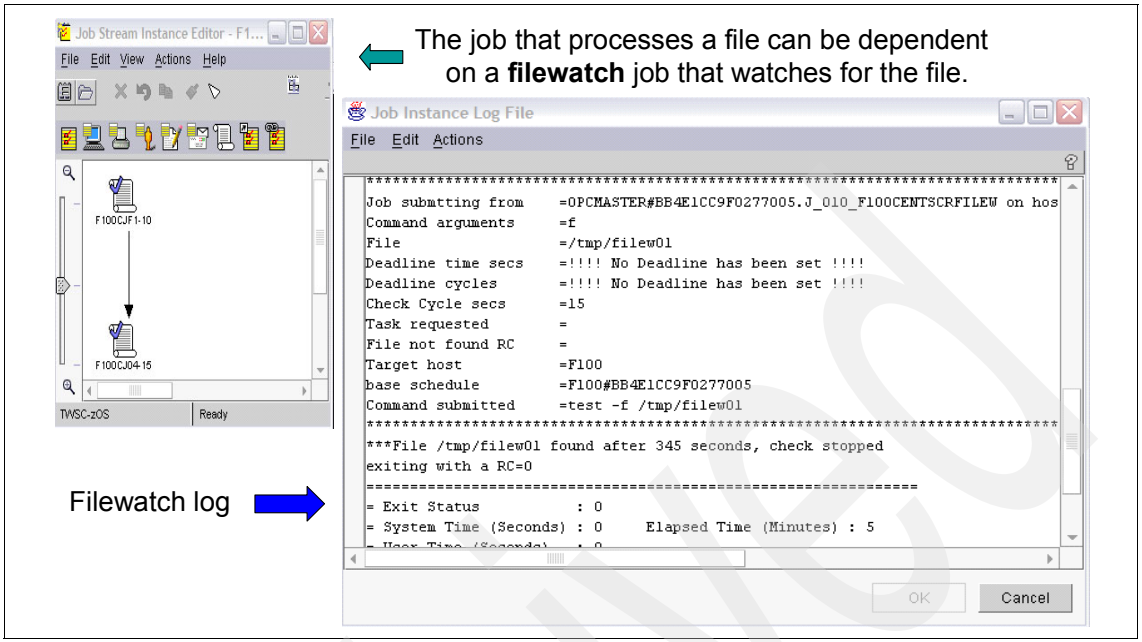


Figure 17-21 How to use filewatch.sh to set up a file dependency

To learn more about filewatch and how to use it, read the detailed description in the comments at the top of the filewatch.sh script.

17.8.2 Handling offline or unlinked workstations

Tip: If the workstation does not link as it should, the cause can be that the writer process has not initiated correctly or the run number for the Symphony file on the fault-tolerant workstation is not the same as the run number on the master. If you mark the unlinked workstations and right-click, a pop-up menu opens as shown in Figure 17-22. Click **Link** to try to link the workstation.

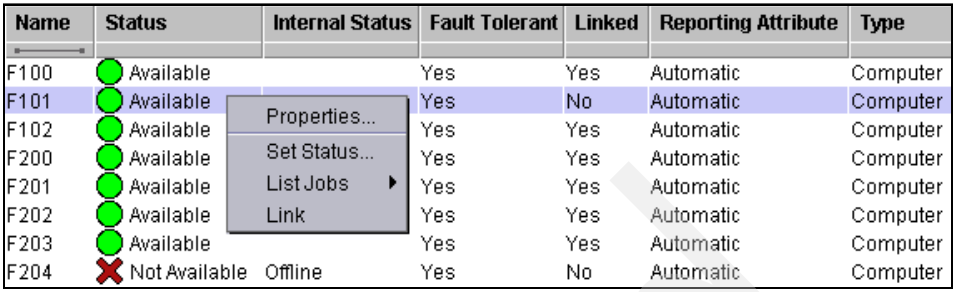


Figure 17-22 Context menu for workstation linking

You can check the Symphony run number and the Symphony status in the ISPF using option 6.6.

Tip: If the workstation is Not Available/Offline, the cause might be that the mailman, batchman, and jobman processes are not started on the fault-tolerant workstation. You can right-click the workstation to open the context menu shown in Figure 17-22, then click **Set Status**. This opens a new window (Figure 17-23), in which you can try to activate the workstation by clicking **Active**. This action attempts to start the mailman, batchman, and jobman processes on the fault-tolerant workstation by issuing a **`conman start`** command on the agent.

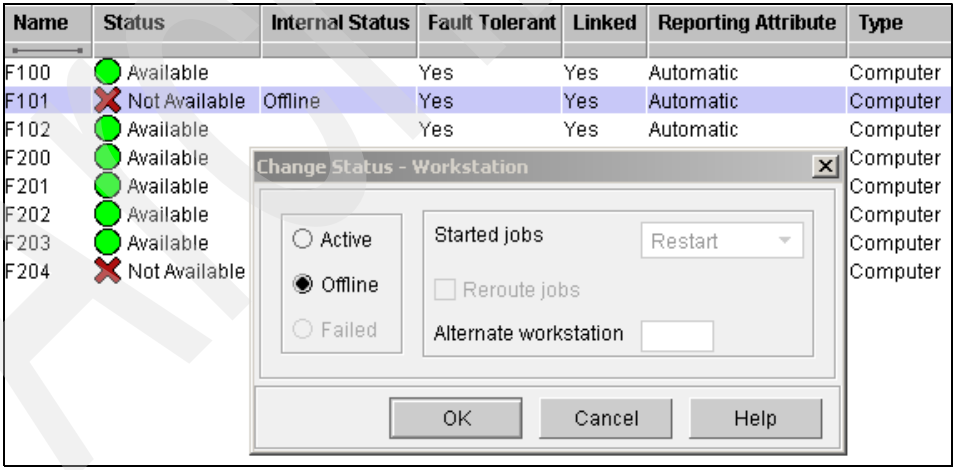


Figure 17-23 Pop-up window to set status of workstation

17.8.3 Using dummy jobs

Because it is not possible to add dependency to the job stream level in Tivoli Workload Scheduler for z/OS (as it is in the Tivoli Workload Scheduler distributed product), dummy start and dummy end general jobs are a workaround for this Tivoli Workload Scheduler for z/OS limitation. When using dummy start and dummy end general jobs, you can always uniquely identify the start point and the end point for the jobs in the job stream.

17.8.4 Placing job scripts in the same directories on FTAs

The SCRPTLIB members can be reused in several job streams and on different fault-tolerant workstations of the same type (such as UNIX or Windows). For example, if a job (script) is scheduled on all of your UNIX systems, you can create one SCRTPLIB member for this job and define it in several job streams on the associated fault-tolerant workstations, though this requires that the script is placed in the same directory on all of your systems. This is another good reason to have all job scripts placed in the same directories across your systems.

17.8.5 Common errors for jobs on fault-tolerant workstations

This section discusses two of the most common errors for jobs on fault-tolerant workstations.

Handling errors in script definitions

When adding a job stream to the current plan in Tivoli Workload Scheduler for z/OS (using JSC or option 5.1 from ISPF), you may see this error message:

```
EQQM071E A JOB definition referenced by this occurrence is wrong
```

This shows that there is an error in the definition for one or more jobs in the job stream and that the job stream is not added to the current plan. If you look in the EQQMLOG for the Tivoli Workload Scheduler for z/OS engine, you will find messages similar to Example 17-21.

Example 17-21 EQQMLOG messages

```
EQQM992E WRONG JOB DEFINITION FOR THE FOLLOWING OCCURRENCE:
EQQZ068E JOBRC IS AN UNKNOWN COMMAND AND WILL NOT BE PROCESSED
EQQZ068I FURTHER STATEMENT PROCESSING IS STOPPED
```

In our example, the F100J011 member in EQQSCLIB looks like:

```
JOBRC JOBSCR('/tivoli/TWS/scripts/japjob1') JOBUSR(tws-e)
```

Note the typo error: JOBRC should be JOBREC. The solution to this problem is simply to correct the error and try to add the job stream again. The job stream must be added to the Tivoli Workload Scheduler for z/OS plan again, because the job stream was not added the first time (due to the typo).

Note: You will get similar error messages in the EQQMLOG for the plan programs if the job stream is added during plan extension. The error messages that are issued by the plan program are:

```
EQQZ068E JOBRC IS AN UNKNOWN COMMAND AND WILL NOT BE PROCESSED
EQQZ068I FURTHER STATEMENT PROCESSING IS STOPPED
EQQ3077W BAD MEMBER F100J011 CONTENTS IN EQQSCLIB
```

Note that the plan extension program will end with return code 0.

If an FTA job is defined in Tivoli Workload Scheduler for z/OS but the corresponding JOBREC is missing, the job will be added to the Symphony file but it will be set to priority 0 and state FAIL. This combination of priority and state is not likely to occur normally, so if you see a job like this, you can assume that the problem is that the JOBREC was missing when the Symphony file was built.

Another common error is a misspelled name for the script or the user (in the JOBREC, JOBSCR, or JOBUSR definition) in the FTA job.

Say we have the JOBREC definition in Example 17-22.

Example 17-22 Typo in JOBREC

```
/* Definiton for F100J010 job to be executed on F100 machine        */
/*                                                                  */
JOBREC JOBSCR('/tivoli/TWS/scripts/jabjob1') JOBUSR(tws-e)
```

Here the typo error is in the name of the script. It should be japjob1 instead of jabjob1. This typo will result in an error with the error code FAIL when the job is run. The error will not be caught by the plan programs or when you add the job stream to the plan in Tivoli Workload Scheduler for z/OS.

It is easy to correct this error using the following steps:

1. Correct the typo in the member in the SCRPTLIB.
2. Add the same job stream again to the plan in Tivoli Workload Scheduler for z/OS.

This way of handling typo errors in the JOBREC definitions is actually the same as if you performed a *rerun from* on a Tivoli Workload Scheduler master. The job stream must be re-added to the Tivoli Workload Scheduler for z/OS plan to have

Tivoli Workload Scheduler for z/OS send the new JOBREC definition to the fault-tolerant workstation agent. Remember, when doing extend or replan of the Tivoli Workload Scheduler for z/OS plan, that the JOBREC definition is built into the Symphony file. By re-adding the job stream we ask Tivoli Workload Scheduler for z/OS to send the re-added job stream, including the new JOBREC definition, to the agent.

Handling the wrong password definition for Windows FTA

If you have defined the wrong password for a Windows user ID in the USRREC topology definition or if the password has been changed on the Windows machine, the FTA job will end with an error and the error code FAIL.

To solve this problem, you have two options:

- Change the wrong USRREC definition and redistribute the Symphony file (using option 3.5 from ISPF).

  This approach can be disruptive if you are running a huge batch load on FTAs and are in the middle of a batch peak.

- Log into the first-level domain manager (the domain manager directly connected to the Tivoli Workload Scheduler for z/OS server; if there is more than one first-level domain manager, log on the first-level domain manager that is in the hierarchy of the FTA), then alter the password either using conman or using a JSC instance pointing to the first-level domain manager. When you have changed the password, simply rerun the job that was in error.

  The USRREC definition should still be corrected so it will take effect the next time the Symphony file is created.

17.8.6 Problems with port numbers

There are two different parameters named PORTNUMBER, one in the SERVOPTS that is used for the JSC and OPC connector, and one in the TOPOLOGY parameters that is used by the E2E Server to communicate with the distributed FTAs.

The two PORTNUMBER parameters must have different values. The localopts for the FTA has a parameter named nm port, which is the port on which netman listens. The nm port value must match the CPUREC CPUTCPIP value for each FTA. There is no requirement that CPUTCPIP must match the TOPOLOGY PORTNUMBER. The value of the TOPOLOGY PORTNUMBER and the HOSTNAME value are imbedded in the Symphony file, which enables the FTA to know how to communicate back to OPCMASTER. The next sections illustrate different ways in which setting the values for PORTNUMBER and CPUTCPIP incorrectly can cause problems in the E2E environment.

Mismatch between CPUTCPIP and nm port

The value for CPUTCPIP in an FTA's CPUREC should always be set to the same port number as the one specified by nm port in the localopts file on that FTA.

We did some tests to see what errors occur if the wrong value is used for CPUTCPIP. In the first test, nm port for the domain manager (DM) HR82 was 31111 but CPUTCPIP was set to 31122, a value that was not used by any FTA on our network. The current plan (CP) was extended to distribute a Symphony file with the wrong CPUTCPIP in place. The domain manager failed to link and the messages in Example 17-23 were seen in the USS stdlist TWSMERGE log.

Example 17-23 Excerpt from TWSMERGE log

```
MAILMAN:+ AWSBCV082I Cpu HR82, Message: AWSDEB003I Writing socket: EDC8128I
Connection refused.
MAILMAN:+ AWSBCV035W WARNING: Linking to HR82 failed, will write to POBOX.
```

Therefore, if the domain manager will not link and the messages shown above are seen in TWSMERGE, the nm port value should be checked and compared to the CPUTCPIP value. In this case, correcting the CPUTCPIP value and running a Symphony Renew job eliminated the problem.

We did another test with the same domain manager, this time setting CPUTCPIP to 31113:

```
CPUREC   CPUNAME(HR82)
   CPUTCPIP(31113)
```

The TOPOLOGY PORTNUMBER was also set to 31113, its normal value:

```
TOPOLOGY PORTNUMBER(31113)
```

After cycling the E2E Server and running a CP EXTEND, the domain manager and all FTAs were LINKED and ACTIVE, which was not expected (Example 17-24).

Example 17-24 Messages showing domain manager and all the FTAs are LINKED and ACTIVE

```
EQQMWSLL -------- MODIFYING WORK STATIONS IN THE CURRENT PLAN  Row 1 to 8 of 8

Enter the row command S to select a work station for modification, or
I to browse system information for the destination.

Row Work station                  L S T R Completed     Active Remaining
cmd name text                             oper  dur.    oper   oper  dur.
'   HR82 PDM on HORRIBLE          L A C A     4    0.00      0    13    0.05
'   OP82 MVS XAGENT on HORRIBLE   L A C A     0    0.00      0     0    0.00
'   R3X1 SAP XAGENT on HORRIBLE   L A C A     0    0.00      0     0    0
```

How could the domain manager be ACTIVE if the CPUTCPIP value was intentionally set to the wrong value? We found that there was an FTA on the network that was set up with nm port=31113. It was actually an MDM (master domain manager) for a Tivoli Workload Scheduler 8.1 distributed-only (not E2E) environment, so our Version 8.2 E2E environment connected to the Version 8.1 master domain manager as if it were HR82. This illustrates that extreme care must be taken to code the CPUTCPIP values correctly, especially if there are multiple Tivoli Workload Scheduler environments present (for example, a test system and a production system).

The localopts nm *ipvalidate* parameter could be used to prevent the overwrite of the Symphony file due to incorrect parameters being set up. If the following line is specified in localopts, the connection would not be allowed if IP validation fails:

```
nm ipvalidate=full
```

However, if SSL is active, the recommendation is to use the following localopts parameter:

```
nm ipvalidate=none
```

PORTNUMBER set to a port that is reserved for another task

We wanted to test the effect of setting the TOPOLOGY PORTNUMBER parameter to a port that is reserved for use by another task. The data set specified by the PROFILE DD statement in the TCPIP statement had the parameters in Example 17-25.

Example 17-25 TOPOLOGY PORTNUMBER parameter

```
PORT
   3000 TCP CICSTCP            ; CICS Socket
```

After setting PORTNUMBER in TOPOLOGY to 3000 and running a CP EXTEND to create a new Symphony file, there were no obvious indications in the messages that there was a problem with the PORTNUMBER setting. However, the following messages appeared in the NETMAN log in USS stdlist/logs:

```
NETMAN:Listening on 3000 timeout 10 started Sun Aug  1 21:01:57 2004
```

These messages then occurred repeatedly in the NETMAN log (Example 17-26).

Example 17-26 Excerpt from the NETMAN log

```
NETMAN:+ AWSEDW02OE Error opening IPC:
NETMAN:AWSDEB001I Getting a new socket: 7
```

If these messages are seen and the domain manager will not link, the following command can be issued to determine that the problem is a reserved TCP/IP port:

```
TSO NETSTAT PORTLIST
```

The output (Example 17-27) shows the values for the PORTNUMBER port (3000).

Example 17-27 Excerpt from the NETMAN log

```
EZZ2350I MVS TCP/IP NETSTAT CS V1R5       TCPIP Name: TCPIP
EZZ2795I Port# Prot User     Flags    Range       IP Address
EZZ2796I ----- ---- ----     -----    -----       ----------
EZZ2797I 03000 TCP  CICSTCP  DA
```

PORTNUMBER set to a port that is already in use

PORTNUMBER in TOPOLOGY was set to 424, which was already in use as the TCPIPPORT by the Controller. Everything worked correctly, but when the E2E Server was shut down, the message in Example 17-28 occurred in the Controller EQQMLOG every 10 minutes.

Example 17-28 Excerpt from the Controller EQQMLOG

```
08/01 18.48.49 EQQTT11E AN UNDEFINED TRACKER AT IP ADDRESS 9.48.204.143
ATTEMPTED TO CONNECT TO THE
08/01 18.48.49 EQQTT11I CONTROLLER. THE REQUEST IS NOT ACCEPTED
EQQMA11E  Cannot allocate connection
EQQMA17E  TCP/IP socket I/O error during Connect() call for
"SocketImpl<Binding=/192.227.118.43,port=31111,localport=32799>",
failed
with error: 146=Connection refused
```

When the E2E Server was up, it handled port 424. When the E2E Server was down, port 424 was handled by the Controller task (which still had TCPIPPORT set to the default value of 424). Because there were some TCP/IP connected trackers defined on that system, message EQQTT11E was issued when the FTA IP addresses did not match the TCP/IP addresses in the ROUTOPTS parameter.

TOPOLOGY PORTNUMBER set the same as SERVOPTS PORTNUMBER

The PORTNUMBER option in SERVOPTS is used for JSC and OPC connector. The PORTNUMBER option in TOPOLOGY is used by the end-to-end-server. If both of these two PORTNUMBER options are set to the same value, E2E processing will still work, but errors will occur when starting the OPC connector.

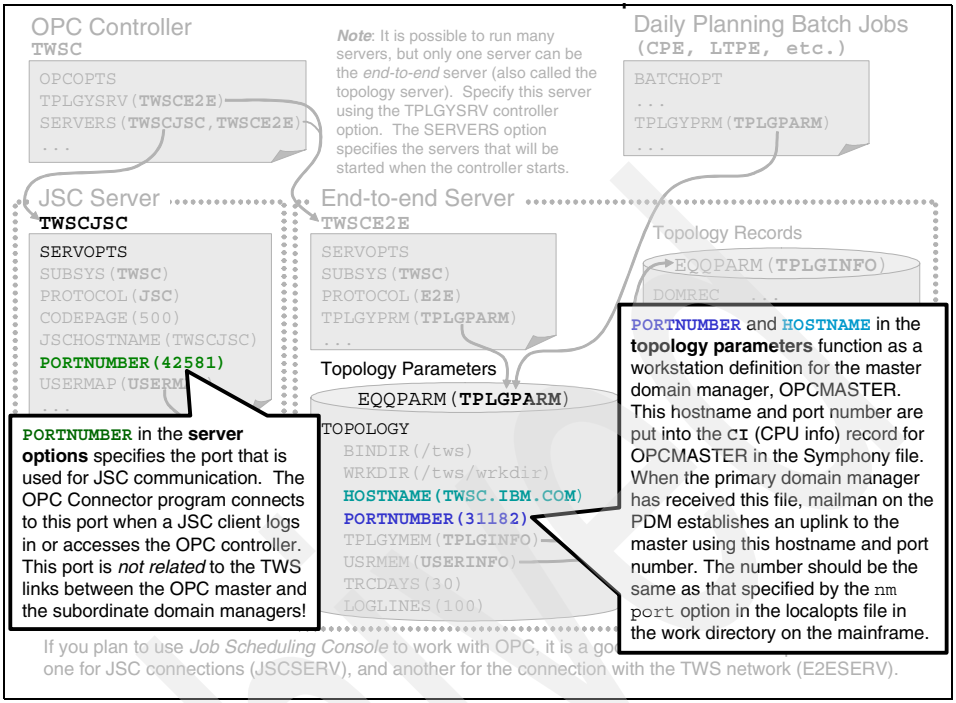


Figure 17-24 The two different PORTNUMBER options, and what they mean

We did a test with the parmlib member for the E2E Server containing the values shown in Example 17-29.

Example 17-29 TOPOLOGY and SERVOPTS PORTNUMBER are the same

```
SERVOPTS   SUBSYS(082C)
   PROTOCOL(E2E,JSC)
   PORTNUMBER(446)
TOPOLOGY PORTNUMBER(446)
```

The OPC connector got the error messages shown in Example 17-30, and the JSC would not function.

Example 17-30 Error message for the OPC connector

```
GJS0005E Cannot load workstation list.  Reason: EQQMA11E Cannot
allocate connection
EQQMA17E TCP/IP socket I/O error during Recv() call for
"Socketlmpl<Binding= dns name/ip address,port=446,localport=4699>"
failed with error" 10054=Connection reset by peer
```

For the OPC connector and JSC to work again, it was necessary to change the TOPOLOGY PORTNUMBER to a different value (not equal to the SERVOPTS PORTNUMBER) and cycle the E2E Server task. Note that this problem could occur if the JSC and E2E PROTOCOL functions were implemented in separate tasks (one task E2E only, one task JSC only) if the two PORTNUMBER values were set to the same value.

If the two PORTNUMBER options were set to the same value, but the JSC server were started before the end-to-end server, it's likely that the JSC server would start up as expected and the end-to-end server would instead generate an error similar to the one above. Again, you must ensure that these two PORTNUMBER options are *not* set to the same value; otherwise, conflicts will arise.

17.8.7 Cannot switch to new Symphony file (EQQPT52E) messages

The EQQPT52E message, with text as shown in Example 17-31, can be a difficult one for troubleshooting because of several different possible causes.

Example 17-31 EQQPT52E message

```
EQQPT52E Cannot switch to the new symphony file:
run numbers of Symphony (x) and CP (y) aren't matching
```

The `x` and `y` in the example message would be replaced by the actual run number values. Sometimes the problem is resolved by running a Symphony Renew or CP REPLAN (or CP EXTEND) job. However, there are some other things to check if this does not correct the problem:

- The EQQPT52E message can be caused if new FTA workstations are added via the Tivoli Workload Scheduler for z/OS dialog, but the TOPOLOGY parms are not updated with the new CPUREC information. In this case, adding the TOPOLOGY information and running a CP batch job should resolve the problem.
- EQQPT52E can also occur if there are problems with the ID used to run the CP batch job or the E2E Server task. One clue that a user ID problem is involved with the EQQPT52E message is if, after the CP batch job completes, there is still a file in the WRKDIR whose name is `Sym` plus the user ID that the CP batch job runs under. For example, if the CP EXTEND job runs under ID TWSRES9, the file in the WRKDIR would be named SymTWSRES9. If security were set up correctly, the SymTWSRES9 file would have been renamed to Symnew before the CP batch job ended.

If the cause of the EQQPT52E still cannot be determined, add the DIAGNOSE statements in Example 17-32 on page 607 to the parm file indicated.

Example 17-32 DIAGNOSE statements added

```
(1) CONTROLLER: DIAGNOSE NMMFLAGS('00003000')
(2) BATCH (CP EXTEND): DIAGNOSE PLANMGRFLAGS('00040000')
(3) SERVER : DIAGNOSE TPLGYFLAGS(X'181F0000')
```

Then collect this list of documentation for analysis.

- Controller and server EQQMLOGs
- Output of the CP EXTEND (EQQDNTOP) job
- EQQTWSIN and EQQTWSOU files
- USS stdlist/logs directory (or a tar backup of the entire WRKDIR

18

End-to-end scheduling troubleshooting

In this chapter, we discuss the troubleshooting for Tivoli Workload Scheduler for z/OS end-to-end scheduling. Apart from listing the most common problems you might encounter and their resolutions, we also want to make you familiar with where to find related information and messages related to troubleshooting. This information is important if you query the IBM Support database or open a PMR.

We cover the following topics in this chapter:

- End-to-end scheduling installation
- Security issues with end-to-end feature
- End-to-end scheduling PORTNUMBER and CPUTCPIP
- E2E Symphony switch and distribution (daily planning jobs)
- OMVS limit problems
- Problems with jobs running on FTAs
- SMP/E maintenance issues
- Other end-to-end scheduling problems
- Other useful end-to-end scheduling information
- Where to find messages in UNIX System Services
- Where to find messages in an end-to-end environment

18.1 End-to-end scheduling installation

In this section, we discuss troubleshooting issues that arise from the installation jobs for end-to-end feature.

18.1.1 EQQISMKD

The EQQISMKD job invokes the EQQMKDIR REXX exec to create directories needed before the SMP/E APPLY job (EQQAPPLE) is run. The EQQAPPLE job populates the binary files in the eqqBINDIR directory. A path prefix, -PathPrefix-, is specified in the EQQISMKD job, and an installation directory (idir) is specified in the EQQMKDIR EXEC. As distributed, the idir is set as follows:

```
idir='usr/lpp/TWS/'
```

If the pathprefix in EQQISMKD is set to / (forward slash), for example, `EQQMKDIR /`, the resulting directory for the binaries would be:

```
/usr/lpp/TWS
```

If instead, a directory of /SERVICE/usr/lpp/TWS was needed, the path prefix in EQQISMKD should be specified as:

```
EQQMKDIR /SERVICE
```

The most common error with the EQQISMKD job is caused by ignoring the following warning in the comments of the job JCL:

```
Ensure the directory specified by -PathPrefix-/usr/lpp exists
 prior to running this job.
```

Example 18-1 shows an example of this type of problem. We did a test using a directory that did not exist (/twstest) as the path prefix for EQQISMKD.

Example 18-1 Using a directory that did not exist as the path prefix for EQQISMKD

```
//SYSTSIN DD *
PROF MSGID
EQQMKDIR /twstest
/*
```

Example 18-2 on page 611 shows the output this produced.

Example 18-2 Output of the EQQISMKD job

```
EQQMKDIR /twstest
The EXEC to create the directories has begun.
It will run for a couple of minutes.
The EQQMKDIR EXEC ran at  23:16:12  on  31 Jul 2004

Directory /twstest/ does not exist.

Please create this directory and resubmit.
```

We corrected this error by using the MKDIR command to create the directory:

```
TWSRES9 @ SC63:/>mkdir /twstest
TWSRES9 @ SC63:/>
```

We ran the EQQISMKD job again, but still got RC=12 and the output shown in Example 18-3.

Example 18-3 Output of the EQQISMKD job after correcting the error

```
EQQMKDIR /twstest
The EXEC to create the directories has begun.
It will run for a couple of minutes.
The EQQMKDIR EXEC ran at  23:18:35  on  31 Jul 2004


 Created the following directories:
 ==================================
No directories were created

 Following directories already exist:
 ====================================
No directories already existed

 Problems creating following directories:
 ========================================
/twstest/usr/lpp/TWS/
  Not created. RC=81    RSN=594003D
/twstest/usr/lpp/TWS/V8R2M0


  Not created. RC=81    RSN=594003D
/twstest/usr/lpp/TWS/V8R2M0/bin
  Not created. RC=81    RSN=594003D
/twstest/usr/lpp/TWS/V8R2M0/bin/IBM
```

```
  Not created. RC=81    RSN=594003D
/twstest/usr/lpp/TWS/V8R2M0/catalog

 Additional messages:
 ====================
No additional messages


Please refer to the OS/390 UNIX Messages and Codes book to interpret
the Return and Reason Codes.
Please correct and resubmit.
The EQQMKDIR EXEC has completed with Return Code 12
```

An easier way to find the meaning of the RSN code 594003D is to use the following command:

```
tso bpxmtext 594003D
```

Example 18-4 shows the output of this command.

Example 18-4 Output of the bpxmtext 594003D command

```
BPXFVLKP 01/16/04
 JRDirNotFound: A directory in the pathname was not found

 ***
 Action: One of the directories specified was not found.  Verify that
the name specified is spelled correctly.
```

Next, we defined the directories up to the lpp subdirectory level, as indicated in the comments of the EQQISMKD job. After resubmitting the job, we got RC=0.

Example 18-5 Defining subdirectories to prevent the 594003D error

```
TWSRES9 @ SC63:/u/twsres9>cd /twstest
TWSRES9 @ SC63:/twstest>mkdir usr
TWSRES9 @ SC63:/twstest>cd /twstest/usr
TWSRES9 @ SC63:/twstest/usr>mkdir lpp
TWSRES9 @ SC63:/twstest/usr>cd /twstest/usr/lpp
TWSRES9 @ SC63:/twstest/usr/lpp>
```

18.1.2 EQQDDDEF

The EQQDDDEF job is run before the SMP/E APPLY job. Example 18-6 shows a note in the comments in the JCL for this job that is important.

Example 18-6 Output of the EQQDDDEF job: note in the comments

```
IN STEP DEFPATH, CHANGE THE -PathPrefix- TO THE
    APPROPRIATE HIGH LEVEL DIRECTORY NAME. THE -PathPrefix-
    STRING MUST MATCH THE SPECIFICATION FOR -PathPrefix-
    STRING IN THE EQQISMKD JOB.

    If no change is performed for the -PathPrefix- string,
    then -PathPrefix- will become the high level directory
    name, which is probably not what you want.
```

This note refers to the part of the EQQDDDEF job shown in Example 18-7.

Example 18-7 Part of the EQQDDDEF job

```
CHANGE PATH('/usr/lpp/TWS/'*,
            '-PathPrefix-/usr/lpp/TWS/'*).
```

What is not entirely clear is that if -PathPrefix- is set to a slash (/) in the EQQISMKD job, in the EQQDDDEF job, -PathPrefix- must be changed to nulls in order to get the command shown in Example 18-8.

Example 18-8 Part of the EQQDDDEF job

```
CHANGE PATH('/usr/lpp/TWS/'*,
            '/usr/lpp/TWS/'*).
```

If -PathPrefix- in EQQISMKD were set to /SERVICE, in EQQDDEF, -PathPrefix- would also be changed to /SERVICE to get the command shown in Example 18-9.

Example 18-9 Part of the EQQDDDEF job

```
CHANGE PATH('/usr/lpp/TWS/'*,
            '/SERVICE/usr/lpp/TWS/'*).
```

18.1.3 EQQPCS05

The EQQPCS05 job creates the work directory (eqqWRKDIR) and creates some of the files that reside in this directory. The security considerations involved in the setup of the EQQPCS05 job are discussed in 18.2, “Security issues with

end-to-end feature" on page 616. Example 18-10 shows the key part of the EQQPCS05 job that must be customized.

Example 18-10 The key part of the EQQPCS05 job that must be customized

```
//STDIN  DD  PATH='/twstest/usr/lpp/TWS/V8R2M0/bin/config',
//           PATHOPTS=(ORDONLY)
//STDENV DD *
eqqBINDIR=/twstest/usr/lpp/TWS/V8R2M0
eqqWRKDIR=/tws/twsae2ew
eqqGID=TWSRES9
eqqUID=TWS810
```

The PATH and eqqBINDIR both refer to the eqqBINDIR directory created by the EQQISMKD job (see 18.1.1, "EQQISMKD" on page 610). The eqqWRKDIR is the name of the work directory. eqqGID and eqqUID are discussed in 18.2, "Security issues with end-to-end feature" on page 616.

After a correct run of the EQQPCS05 job, the contents of the work directory (from the z/OS UNIX System Services command **ls -la**) should look similar to the contents shown in Example 18-11.

Example 18-11 Contents of the work directory after a correct run of EQQPCS05

```
TWSRES9 @ SC63:/tws/twsae2ewz>ls -la
total 502988
drwxrwxrwx   5 TWSRES9  TWS810      832 Jul 30 05:16 .
drwxr-xr-x 16 HAIMO    SYS1       8192 Jul  6 18:39 ..
-rw-------   1 TWSRES9  TWS810      128 Jul 17 22:35 Intercom.msg
-rw-------   1 TWSRES9  TWS810       48 Jul 17 22:35 Mailbox.msg
-rw-rw----   1 TWSRES9  TWS810      839 Jul 31 23:48 NetConf
-rw-rw----   1 TWSRES9  TWS810       48 Jul 17 22:36 NetReq.msg
-rw-------   1 TWSRES9  TWS810   128672256 Jul 17 22:34 Sinfonia
-rw-r--r--   1 TWSRES9  TWS810   128672256 Jul 17 22:35 Symphony
-rw-rw----   1 TWSRES9  TWS810      1118 Jul 31 23:48
TWSCCLog.properties
-rw-rw-rw-   1 TWSRES9  TWS810      720 Jul 17 22:36 Translator.chk
-rw-rw-rw-   1 TWSRES9  TWS810        0 Jul 17 21:22 Translator.wjl
-rw-rw----   1 TWSRES9  TWS810     2743 Jul 31 23:48 localopts
drwxrwx---   2 TWSRES9  TWS810      544 Jul 17 22:33 mozart
drwxrwx---   2 TWSRES9  TWS810      384 Jul 17 22:33 pobox
drwxrwxr-x   4 TWSRES9  TWS810       384 Jul 17 21:21 stdlist
```

One problem not related to security that we have seen in testing occurs if the work directory mount point is not mounted at the time that the EQQPCS05 job is run. Example 18-12 shows the error messages seen in this case.

Example 18-12 Error messages

```
You are running as TWSRES9 (68248)
Running configuration with uid=68248
Configuration is running with real userid TWSRES9 (68248)
mkdir: FSUM6404 directory "/tws/twsae2ewz/mozart": EDC5111I Permission
denied.
 Configure failed executing: mkdir /tws/twsae2ewz/mozart.
Please correct the problem and rerun configure.
```

If for some reason it is necessary to delete and re-create the work directory (WRKDIR), issue the following command from TSO z/OS UNIX System Services to delete the WRKDIR first, and then run the EQQPCS05 job:

```
rm -r /WRKDIR
```

So, if for example WRKDIR is defined as /var/TWS/inst, use this procedure:

1. Issue the following z/OS UNIX System Services command:

   ```
   rm -r /var/TWS/inst
   ```

2. Run the EQQPCS05 job.

18.1.4 EQQPH35E message after applying or installing maintenance

If authorization is lost for the files in the eqqBINDIR directory, message EQQPH35E might be seen. Using a utility other than DFDSS backup/restore to move HFS data sets is the most common reason for losing APF authorization in the bin directory. This is explained in DOC APAR PQ77535. The files in the eqqBINDIR/bin and eqqBINDIR/bin/IBM directories should have the following attributes:

```
APF authorized . . . : 1
```

If this is not the case, the following command can be issued to modify the APF attribute for the eqqBINDIR/bin and eqqBINDIR/bin/IBM directories:

```
extattr +a /directory-name
```

Another possible cause of message EQQPH35E is an HFS mount problem. The following message can appear in the E2E Server EQQMLOG:

```
EQQPH35E CANNOT START STARTER PROCESS: BPX1ATX FAILED WITHRC=157,
RSN=5B4B000D
```

The domain manager (DM) fails to connect, and cycling the E2E Server or the domain manager has no affect on the problem. The 5B4B000D reason code means that the ROOT file system (HFS) is mounted on more than one LPAR for read/write access. This is not allowed, and likely is caused by an incorrect GRS configuration. When the HFS problem is resolved, E2E Server will restart correctly.

18.2 Security issues with end-to-end feature

There probably has been more confusion about the comments in the EQQPCS05 job, as shown in Example 18-13, than anything else related to E2E.

Example 18-13 Comments in the EQQPCS05 job

```
This JCL must be executed by a user:
 a) with USS uid equal to 0
 b) with an user permitted to the BPX.SUPERUSER FACILITY class
    profile within RACF
 c) or with the user specified below in eqqUID and belonging on
    the group specified in eqqGID
```

These statements are true. However, they have been misread in many cases to mean that root authority is needed to run EQQPCS05, or for the eqqUID itself. This is untrue. Not only is root authority not needed; to use a root ID (UID(0)) either to run the EQQPCS05 job or as eqqUID causes more problems than if a non-root ID is used.

Here are the key points concerning EQQPCS05 and end-to-end security:

- eqqUID must be the user ID of the E2E Server task.
- eqqGID must be a group to which eqqUID belongs.
- The user ID of the Controller task must belong to eqqGID.
- The user ID of all CP batch jobs (EXTEND, REPLAN, and SYMPHONY RENEW) must belong to eqqGID.
- The user ID of any Tivoli Workload Scheduler for z/OS dialog user who is allowed to submit CP BATCH jobs from the dialog must belong to eqqGID.
- The user ID under which the EQQPCS05 job runs must belong to eqqGID.
- All user IDs listed above must have a default group (DFLTGRP) that has a valid OMVS segment (that is, a GID is assigned to that group).
- If a non-unique UID is used for eqqUID (that is, a root UID or a non-root UID that is assigned to multiple user IDs), *all* user IDs sharing that UID must have a DFLTGRP that has a valid OMVS segment.

The reason that the non-unique user IDs can cause a problem is that RACF maintains a list in memory that relates UIDs to user IDs. For any UID, there can be only one user ID on the list, and if the UID is non-unique, the user ID will vary depending on the last user ID that was referenced for that UID. If a **getuid** command is issued to obtain the UID/user ID information, and the current user ID on the list has a default group (DFLTGRP) that does not have a valid OMVS segment, the **getuid** command will fail, causing end-to-end processing to fail.

In the following sections, we describe some examples of common security-related problems in end-to-end.

18.2.1 Duplicate UID

The messages, as shown in Example 18-14, in the EQQMLOG of the E2E Server might indicate a problem with a duplicate UIDs (two or more user IDs with the same UID).

Example 18-14 Messages in the EQQMLOG related to duplicate UIDs

```
EQQPT60E An END-TO-END process could end abnormally
EQQPT09E The Netman process (pid=nnnnn) ended abnormally
```

The messages shown in Example 18-15 would also be seen in the USS netman log with duplicate UIDs.

Example 18-15 Excerpt from the netman log

```
NETMAN:+ AWSEDW026E Error firing child process: Service 2002 for
NETMAN:+ TWS_z/OS/8.2 on OPCMASTER(UNIX)
AWSDCJ01OE Error opening new NETMAN:+ stdlist, Error: EDC5123I Is a
directory.
```

In the case that caused these error messages, there were two user IDs, TSTCOPC and TSUOMVS, that both were defined with UID(3). TSTCOPC was used as eqqUID when the EQQPCS05 job was run.

Example 18-16 TSTCOPC and TSUOMVS user IDs

```
//STDENV DD *
eqqBINDIR=/usr/lpp/TWS/V8R2M0
eqqWRKDIR=/BAYP/twswork
eqqGID=OPCGRP
eqqUID=TSTCOPC
```

However, the ID TSUOMVS was the ID under which the EQQPCS05 job was run. The E2E Server and Controller tasks both ran under the TSTCOPC ID. A display

of the stdlist and logs directories (Example 18-17) shows that TSUOMVS was the owner when the owner should have been TSTCOPC (eqqUID).

Example 18-17 Display of the stdlist and logs directories

```
/BAYP/twswork/stdlist
# ls -lisa
total 72
3121 16 drwxrwxr-x 4 TSUOMVS OPCGRP 8192 Feb 18 14:50 .
0 16 drwxrwxrwx 5 TSUOMVS OPCGRP 8192 Feb 18 17:31 ..
3125 16 drwxrwxrwx 2 TSUOMVS OPCGRP 8192 Feb 18 14:50 2004.02.18
3122 16 drwxrwxr-x 2 TSUOMVS OPCGRP 8192 Feb 18 14:50 logs
3124 8 -rwxrwxrwx 1 TSUOMVS OPCGRP 32 Feb 18 16:24 stderr
3123 0 -rwxrwxrwx 1 TSUOMVS OPCGRP 0 Feb 18 16:24 stdout

/BAYP/twswork/stdlist/logs
# ls -lisa
total 80
3122 16 drwxrwxr-x 2 TSUOMVS OPCGRP 8192 Feb 18 14:50 .
3121 16 drwxrwxr-x 4 TSUOMVS OPCGRP 8192 Feb 18 14:50 ..
3127 8 -rwxr--r-- 1 TSUOMVS OPCGRP 3450 Feb 18 16:29
20040218_E2EMERGE.log
3129 40 -rwxr--r-- 1 TSUOMVS OPCGRP 17689 Feb 18 17:35
20040218_NETMAN.log
```

The way permissions are set up on the logs, only the owner or a root ID can write to them, and because the ownership shows up as TSUOMVS while the E2E Server is running under ID TSTCOPC (and thus the USS netman process is running under TSTCOPC), the netman process is unable to write to the logs directory, which causes the AWSEDW026E error in the netman log.

To avoid this problem, ensure that the eqqUID ID used in the EQQPCS05 job has a unique OMVS UID. This can be enforced by implementing the support for RACF APAR OW52135 (NEW SUPPORT FOR THE MANAGEMENT OF UNIX UIDS AND GIDS) or by careful assignment of UIDs when RACF IDs are created. Using a root authority ID, it is possible to display the current list of IDs and UIDs from the TSO ISHELL environment as follows:

1. From TSO ISHELL, select **SETUP** with your cursor, and then enter 7 (ENABLE SU MODE).
2. Select **SETUP** with your cursor again, and then enter 2 (USER LIST).
3. In the USER LIST display, select **FILE** with your cursor, and then enter 2 (SORT UID).
4. Select a UID that is not currently in use.

18.2.2 E2E Server user ID not eqqUID

To document the effect of running the E2E Server with a user ID that is not eqqUID, we ran the EQQPCS05 job with eqquid=O82STSO and then set up the E2E Server task O82S to run under ID O82C using the RACF commands listed in Example 18-18.

Example 18-18 RACF commands

```
RALTER  STARTED O82S.* -
        STDATA(USER(O82C) GROUP(OPC) TRUSTED(NO))
SETROPTS RACLIST(STARTED) REFRESH
SETROPTS GENERIC(STARTED) REFRESH
```

The RACF error messages shown in Example 18-19 were issued for the O82S (E2E Server) task; however, the task kept running.

Example 18-19 RACF error messages

```
ICH408I USER(O82C    ) GROUP(OPC     ) NAME(TWS82 CONTROLLER    )
  /var/TWS820/inst/Intercom.msg
  CL(FSOBJ   ) FID(01D9F0F1F9F1C5000F04000018F90000)
  INSUFFICIENT AUTHORITY TO OPEN
  ACCESS INTENT(RW-)  ACCESS ALLOWED(GROUP      ---)
  EFFECTIVE UID(0000000146)  EFFECTIVE GID(0001000009)
ICH408I USER(O82C    ) GROUP(OPC     ) NAME(TWS82 CONTROLLER    )
  /var/TWS820/inst/Mailbox.msg
  CL(FSOBJ   ) FID(01D9F0F1F9F1C5000F04000018F00000)
  INSUFFICIENT AUTHORITY TO OPEN
  ACCESS INTENT(RW-)  ACCESS ALLOWED(GROUP      ---)
  EFFECTIVE UID(0000000146)  EFFECTIVE GID(0001000009)
ICH408I USER(O82C    ) GROUP(OPC     ) NAME(TWS82 CONTROLLER    )
  /var/TWS820/inst/Intercom.msg
  CL(FSOBJ   ) FID(01D9F0F1F9F1C5000F04000018F90000)
  INSUFFICIENT AUTHORITY TO OPEN
  ACCESS INTENT(RW-)  ACCESS ALLOWED(GROUP      ---)
  EFFECTIVE UID(0000000146)  EFFECTIVE GID(0001000009)
```

The server EQQMLOG contained the messages shown in Example 18-20.

Example 18-20 Server EQQMLOG messages

```
EQQPT35E Unable to Open Symphony and/or events files, Reason:
AWSBDY102E Attempt to use NULL or uninitialized comarea.
 EQQPT40I Output Translator thread is shutting down
```

The user ID of the server task was corrected using the RACF commands shown in Example 18-21, and then the server task was restarted.

Example 18-21 RACF commands to correct the user ID of the server task

```
RALTER  STARTED O82S.* -
        STDATA(USER(O82STSO) GROUP(OPC) TRUSTED(NO))
SETROPTS RACLIST(STARTED) REFRESH
SETROPTS GENERIC(STARTED) REFRESH
```

Now the user ID of the server task matches eqqUID, but access to the stdlist/logs directory in USS WRKDIR was still not right, causing the RACF message shown in Example 18-22 to be issued continuously in the server JESMSGLG.

Example 18-22 Excerpt from the server JESMSGLG

```
ICH408I USER(O82STSO ) GROUP(OPC     ) NAME(O82STSO
    /var/TWS820/inst/stdlist/logs//20040801_E2EMERGE.log
    CL(FSOBJ   ) FID(01D9F0F1F9F1C5000F0400001D360000)
    INSUFFICIENT AUTHORITY TO OPEN
    ACCESS INTENT(-W-)  ACCESS ALLOWED(GROUP      R--)
    EFFECTIVE UID(0000000154)  EFFECTIVE GID(0001000009)
```

The server EQQMLOG still had the following message:

```
EQQPT09E The Netman process (pid=33554521) ended abnormally
```

Example 18-23 shows that the files in the stdlist/logs directory were still owned by the O82C user ID (the user ID that was used the first time the server task was started).

Example 18-23 Files in the stdlist/logs directory

```
-rwxr--r--   1 O82C     OPC        21978 Aug  1 00:48 20040801_E2EMERGE.log
-rwxr--r--   1 O82C     OPC         4364 Aug  1 00:48 20040801_NETMAN.log
SVIOLA:/SYSTEM/var/TWS820/inst/stdlist/logs>
```

We tried to correct this with the **chown** command, but this was unsuccessful. To get the server working correctly again, we deleted the wrkdir (using the command **rm -r /wrkdir**) and then reran the EQQPCS05 job.

18.2.3 CP batch user ID not in eqqGID

If a CP batch job (CP EXTEND, CP REPLAN, or SYMPHONY RENEW) is run under a user ID that is not part of the group eqqGID, the messages shown in

Example 18-24 will appear in the EQQMLOG of the CP batch job, and the job will end with RC=04.

Example 18-24 Excerpt from EQQMLOG

```
EQQ3105I A NEW CURRENT PLAN (NCP) HAS BEEN CREATED
EQQ3053E EQQsyncr : EFFECTIVE USERID NOT GOTTEN
EQQ3099E THE REASON OF THE PREVIOUS ERROR IS:
EQQ3099I EDC5164I SAF/RACF error.
EQQ3088E THE SYMPHONY FILE HAS NOT BEEN CREATED
```

To correct this, either change the user ID of the CP batch job to a user ID that is already part of the group eqqGID, or connect the user ID to the group eqqGID with the following RACF command:

```
connect (userid) group(eqqGID) uacc(update)
```

After this is done, when the CP batch job is run again, the messages shown in Example 18-25 should be issued instead of the ones shown in Example 18-24.

Example 18-25 Excerpt from EQQMLOG

```
EQQ3105I A NEW CURRENT PLAN (NCP) HAS BEEN CREATED
EQQ3106I WAITING FOR SCP
EQQ3107I SCP IS READY: START JOBS ADDITION TO SYMPHONY FILE
EQQ3108I JOBS ADDITION TO SYMPHONY FILE COMPLETED
```

18.2.4 General RACF check procedure for E2E Server

RACF (security) errors are seen when starting the E2E Server task for files in the WRKDIR (work directory) such as /var/TWS/inst/stdlist, shown in Example 18-26.

Example 18-26 RACF (security) errors

```
ICH408I USER(TWSUSR ) GROUP(TWSFG ) NAME(TIVOLI WRKD SCHEDULE) /dev/null CL(DIRSRCH )
FID(01C8C6E2C1F0F141461C0000000000003) INSUFFICIENT AUTHORITY TO OPEN ACCESS
INTENT(--X) ACCESS ALLOWED(OTHER RW-) EFFECTIVE UID(0000000039) EFFECTIVE
GID(0000000018) ICH408I USER(TWSUSR ) GROUP(TWSFG ) NAME(TIVOLI WRKD SCHEDULE)
/var/TWS/TEST/stdlist/stdout CL(DIRSRCH ) FID(01C8C6E2C1F0F141461C0000000000003)
INSUFFICIENT AUTHORITY TO OPEN ACCESS INTENT(--X) ACCESS ALLOWED(OTHER RW-) EFFECTIVE
UID(0000000039) EFFECTIVE GID(0000000018)
```

This can indicate RACF configuration problems or that the PTF for PK11182 is not installed. Problems of this type can be diagnosed with greater ease if the PTFs for PK01415 have been applied. The PTFs are UK04927, UK04908, and UK04925 for Tivoli Workload Scheduler for z/OS 8.2. However, if these PTFS are

applied, the following PTFS must also be applied to avoid problems that *appear* to be security problems but can occur even if there are no problems with the security configuration: UK07035, UK10116, UK10117, and UK10118.

If all of these PTFs are applied, follow this procedure below to verify that the end-to-end configuration is correct from a security standpoint:

1. From the EQQPCS05 job, get the values for eqqUID (the user ID of the E2E Server task) and eqqGID (the GROUP that all user IDs that can modify the Symphony file must belong to).
2. Verify that the values for eqqBINDIR and eqqWRKDIR in the EQQPCS05 job match the BINDIR and WRKDIR parameters respectively in the E2E Server TOPOLOGY parms. If these values do not match, correct the problem and try the E2E Server again. If there are still problems, continue with this procedure.
3. Verify that the E2E Server task is running under ID eqqUID and that eqqUID belongs to group eqqGID. You can do this by issuing this command:

   ```
   LG eqqGID
   ```

4. Check the output of the LG command to verify that eqqUID is in the list of connected user IDs).
5. If the check from step 4 shows that the user ID is as expected, continue with step 6. If this check fails, fix the problem and try the E2E Server task again. If the error still persists, continue with step 6.
6. Issue this command from TSO OMVS, and verify that both the BINDIR and WRKDIR directories are mounted:

   ```
   df -k
   ```

7. If either directory is not mounted, retry the E2E Server task; otherwise continue to step 8.
8. From TSO OMVS, change directory to the WRKDIR and issue this command:

   ```
   ls -la
   ```

   The resulting display should be similar to Example 18-27.

Example 18-27 ls -la output

```
-rw------- 1 O82STSO OMVS 48 Jan 19 15:14 Intercom.msg
-rw------- 1 O82STSO OMVS 48 Jan 19 15:14 Mailbox.msg
-rw-rw---- 1 O82STSO OMVS 839 Oct 11 14:09 NetConf
-rw-rw---- 1 O82STSO OMVS 48 Jan 19 15:03 NetReq.msg
-rw------- 1 O82STSO OMVS 27136 Jan 18 18:09 Sinfold
-rw------- 1 O82STSO OMVS 62976 Jan 19 15:03 Sinfonia
-rw-rw-rw- 1 O82STSO OMVS 0 Jan 18 18:05 Starter.chk
-rw-r--r-- 1 O82STSO OMVS 27136 Jan 19 15:02 Symold
-rw-r--r-- 1 O82STSO OMVS 62976 Jan 19 15:13 Symphony
```

```
-rw-rw---- 1 O82STSO OMVS 4671 Dec 30 16:51 TWSCCLog.properties
-rw-rw-rw- 1 O82STSO OMVS 720 Jan 19 15:03 Translator.chk
-rw-rw-rw- 1 O82STSO OMVS 136 Jan 9 16:10 Translator.wjl
drwxr-xr-x 2 O82STSO OMVS 8192 Oct 11 14:19 ftbox
-rw-rw---- 1 O82STSO OMVS 2744 Oct 11 14:09 localopts
drwxrwx--- 2 O82STSO OMVS 8192 Oct 11 14:19 mozart
drwxrwx--- 2 O82STSO OMVS 8192 Jan 19 15:03 pobox
drwxrwxr-x 12 O82STSO OMVS 8192 Jan 19 15:02 stdlist
```

In this example, O82STSO is eqqUID and OMVS is eqqGID. If eqqUID is a root user (UID=0) you may see *any* other root ID displayed.

9. If the values you see do not agree with the EQQPCS05 job, the EQQPCS05 job might have been run using a different JCL member than the one you assumed was correct. Research this and rerun EQQPCS05 if necessary, then start over from the beginning of this check list. Before running EQQPCS05, you must delete the WRKDIRby issuing this command from a TSO OMVS session:

   ```
   rm -r /WRKDIR
   ```

   If the values shown are the expected ones, continue to step 10.

10. Use the output of the LG eqqGID command from step 3 on page 622 (or simply reissue the command if you no longer have the output) to verify that all of the following IDs are connected to eqqGID:

   – The ID used to run the EQQPCS05 job.
   – The ID of the controller and E2E Server (already checked in step 2).
   – The ID of *any* user that runs a batch daily planning job (CP EXTEND, CP REPLAN, Symphony Renew).
   – The ID of any TSO user who can submit a daily planning job from the TWS dialog.

   If no user IDs are found that should be connected to eqqGID but are not, continue to step 11. If you find any user IDs that should belong to eqqGID but do not, correct this and retry the E2E Server task. If the problem persists, continue with step 11.

11. There may be a problem with duplicate UIDs. Either the UID of eqqUID must be unique or all user IDs that have this UID must have a default group (DFLTGRP) that has an OMVS segment (a GID). You can check this from any TSO ID that has superuser authority (UID=0) using these steps:

   a. Enter TSO ISHELL.
   b. Cursor-select **SETUP** on the top line.
   c. Choose option 2 (USER LIST).

d. Cursor-select **FILE** on the next panel.

e. Choose option 2 (SORT UID).

f. Use PF8 to scroll through the list and determine how many IDs have the same UID as the eqqUID user ID.

12. If there are no duplicate UIDs. continue with step 15. If more than one user ID has UID=eqqUID, issue this command for each user ID and note the DFLTGRP (DEFAULT-GROUP) value:

```
LU XXXXXX
```

13. For each DFLTGRP found, issue this command:

```
LG gggggggg OMVS
```

14. If you find a DFLTGRP that does not have a GID listed under OMVS INFORMATION, you must correct this and retry the E2E Server. If the problem persists, continue with step 15.

15. After you complete this checklist, if the security messages persist in the E2E Server task, a PMR should be opened to pursue the problem further. Mention to Tivoli Workload Scheduler for z/OS that you have followed this checklist so that it can be revised to include whatever undocumented problem resulted in the error.

18.2.5 Security problems with BPX_DEFAULT_USER

If BPX_DEFAULT_USER is not configured correctly, there can be security errors with user IDs for Tivoli Workload Scheduler for z/OS E2E. Setting certain variables such as HOME, PROGRAM, and UID correctly is essential for OMVS access.

The errors in Example 18-28 may be seen when submitting jobs.

Example 18-28 Errors related with security problems

```
EQQ3053E EQQsyncr : EFFECTIVE USERID NOT GOTTEN
EQQ3099E THE REASON OF THE PREVIOUS ERROR IS:
EQQ3099I EDC5164I SAF/RACF error.
```

This indicates a problem with either the user settings or the settings in RACF for BPX.DEFAULT.USER.

The best way to configure user IDs that will be running end-to-end scheduling or submitting jobs with end-to-end scheduling is to ensure that all user IDs have explicitly defined OMVS segments. If that is not possible, the BPX_DEFAULT_USER profile must be correctly defined with valid HOME, PROGRAM, and UID values. A user ID with no explicitly defined OMVS segment

assumes the characteristics defined in BPX.DEFAULT.USER and needs these three variables set in a valid manner so that the user can work as needed for end-to-end scheduling.

In addition, APARs such as PK01415 have tightened security while giving better messages to help diagnose security issues.

18.3 End-to-end scheduling PORTNUMBER and CPUTCPIP

There are two different parameters named PORTNUMBER: one in the SERVOPTS that is used for the JSC and OPC connector, and one in the TOPOLOGY parameters that is used by the E2E Server to communicate with the distributed FTAs. The two PORTNUMBER parameters must have different values. The localopts for the FTA has a parameter named nm port, which is the port on which netman listens. The nm port value must match the CPUREC CPUTCPIP value for each FTA. There is no requirement that CPUTCPIP must match the TOPOLOGY PORTNUMBER. The value of the TOPOLOGY PORTNUMBER and the HOSTNAME value are imbedded in the Symphony file, which allows the FTA to know how to communicate back to OPCMASTER. The next sections illustrate different ways in which incorrectly setting the values for PORTNUMBER and CPUTCPIP can cause problems in the end-to-end scheduling environment.

18.3.1 CPUTCPIP not same as nm port

The value for CPUTCPIP in the CPUREC parameter for an FTA should always be set to the same port that the FTA has defined as nm port in localopts.

We ran tests to see what errors occur if the wrong value is used for CPUTCPIP. In the first test, nm port for the domain manager (DM) HR82 was 31111, but CPUTCPIP was set to 31122, a value that was not used by any FTA on our network. The current plan (CP) was extended to distribute a Symphony file with the wrong CPUTCPIP in place. The domain manager failed to link, and the messages in Example 18-29 were seen in the USS stdlist TWSMERGE log.

Example 18-29 Excerpt from TWSMERGE log

```
MAILMAN:+ AWSBCV082I Cpu HR82, Message: AWSDEB003I Writing socket:
EDC8128I Connection refused.
MAILMAN:+ AWSBCV035W WARNING: Linking to HR82 failed, will write to
POBOX.
```

Therefore, if the DM will not link, and the messages shown in Example 18-29 are seen in TWSMERGE, the nm port value should be checked and compared to the CPUTCPIP value. In this case, correcting the CPUTCPIP value and running a Symphony Renew job eliminated the problem.

We did another test with the same domain manager, this time setting CPUTCPIP to 31113.

Example 18-30 Setting CPUTCPIP to 31113

```
CPUREC   CPUNAME(HR82)
          CPUTCPIP(31113)
```

The TOPOLOGY PORTNUMBER was also set to its normal value of 31113:

```
TOPOLOGY PORTNUMBER(31113)
```

After cycling the E2E Server and running a CP EXTEND, the domain manager and all of the FTAs were linked and active, which is not what was expected.

Example 18-31 Messages showing DM and all the FTAs are LINKED and ACTIVE

```
EQQMWSLL -------- MODIFYING WORK STATIONS IN THE CURRENT PLAN  Row 1 to 8 of 8

Enter the row command S to select a work station for modification, or
I to browse system information for the destination.

Row Work station                 L S T R Completed     Active Remaining
cmd name text                              oper  dur.    oper   oper  dur.
'   HR82 PDM on HORRIBLE         L A C A     4    0.00      0     13    0.05
'   OP82 MVS XAGENT on HORRIBLE  L A C A     0    0.00      0      0    0.00
'   R3X1 SAP XAGENT on HORRIBLE  L A C A     0    0.00      0      0    0
```

How could the DM be active if the CPUTCPIP value was intentionally set to the wrong value? We found that there was an FTA in the network that was set up with nm port=31113. It was actually a master domain manager (MDM) for a Tivoli Workload Scheduler V8.1 Distributed only (not E2E) environment. So our V8.2 end-to-end scheduling environment connected to the V8.1 master domain manager as though it were HR82. This illustrates that extreme care must be taken to code the CPUTCPIP values correctly, especially if there are multiple Tivoli Workload Scheduler environments present (for example, a test system and a production system).

The localopts nm ipvalidate parameter could be used to prevent the overwrite of the Symphony file due to incorrect parameters being set up. If the following value is specified in localopts, the connection would not be allowed if IP validation fails:

```
nm ipvalidate=full
```

However, if SSL is active, we recommend that you use the following localopts parameter:

```
nm ipvalidate=none
```

18.3.2 PORTNUMBER set to PORT reserved for another task

We wanted to test the effect of setting the TOPOLOGY PORTNUMBER parameter to a port that is reserved for use by another task. The data set specified by the PROFILE DD statement in the TCPIP proc had the parameters shown in Example 18-32.

Example 18-32 TOPOLOGY PORTNUMBER parameter

```
PORT
  3000 TCP CICSTCP            ; CICS Socket
```

After setting PORTNUMBER in TOPOLOGY to 3000 and running a CP EXTEND to create a new Symphony file, there were no obvious indications in the messages that there was a problem with the PORTNUMBER setting. However, the following message appeared in the netman log in USS stdlist/logs:

```
NETMAN:Listening on 3000 timeout 10 started Sun Aug  1 21:01:57 2004
```

These messages then occurred repeatedly in the netman log (Example 18-33).

Example 18-33 Excerpt from the netman log

```
NETMAN:+ AWSEDW020E Error opening IPC:
NETMAN:AWSDEB001I Getting a new socket: 7
```

If these messages are seen, and the domain manager will not link, issue this command to determine that the problem is a reserved TCP/IP port:

```
TSO NETSTAT PORTLIST
```

Example 18-34 shows the output with the values for the PORTNUMBER port (3000).

Example 18-34 Excerpt from the netman log

```
EZZ2350I MVS TCP/IP NETSTAT CS V1R5       TCPIP Name: TCPIP
EZZ2795I Port# Prot User     Flags    Range       IP Address
```

```
EZZ2796I ----- ---- ----     -----    -----       ----------
EZZ2797I 03000 TCP  CICSTCP  DA
```

18.3.3 PORTNUMBER set to PORT already in use

PORTNUMBER in TOPOLOGY was set to 424, which was already in use as the TCPIPPORT by the Controller. Everything worked correctly, but when the E2E Server was shut down, the message shown in Example 18-35 occurred in the Controller EQQMLOG every 10 minutes.

Example 18-35 Excerpt from the Controller EQQMLOG

```
08/01 18.48.49 EQQTT11E AN UNDEFINED TRACKER AT IP ADDRESS 9.48.204.143
ATTEMPTED TO CONNECT TO THE
08/01 18.48.49 EQQTT11I CONTROLLER. THE REQUEST IS NOT ACCEPTED
EQQMA11E  Cannot allocate connection
EQQMA17E  TCP/IP socket I/O error during Connect() call for
"SocketImpl<Binding=/192.227.118.43,port=31111,localport=32799>",
failed with error: 146=Connection refused
```

When the E2E Server was up, it handled port 424. When the E2E Server was down, port 424 was handled by the Controller task (which still had TCPIPPORT set to the default value of 424). Because there were some TCP/IP connected trackers defined on that system, message EQQTT11E was issued, because the FTA IP addresses did not match the TCP/IP addresses in the ROUTOPTS parameter.

18.3.4 TOPOLOGY and SERVOPTS PORTNUMBER set to same value

The PORTNUMBER in SERVOPTS is used for JSC and OPC connector. If the TOPOLOGY PORTNUMBER is set to the same value as the SERVOPTS PORTNUMBER (for a server defined with PROTOCOL(E2E,JSC)), end-to-end scheduling processing will still work, but errors will occur when starting the OPC connector. For example, we did a test with the parmlib member for the E2E Server containing the values shown in Example 18-36.

Example 18-36 TOPOLOGY and SERVOPTS PORTNUMBER are the same

```
SERVOPTS   SUBSYS(O82C)
   PROTOCOL(E2E,JSC)
   PORTNUMBER(446)
TOPOLOGY PORTNUMBER(446)
```

The OPC connector got the error messages shown in Example 18-37, and the JSC would not function.

Example 18-37 Error message for the OPC connector

```
GJS0005E Cannot load workstation list.  Reason: EQQMA11E Cannot
allocate connection
EQQMA17E TCP/IP socket I/O error during Recv() call for
"Socketlmpl<Binding= dns name/ip address,port=446,localport=4699>"
failed with error" 10054=Connection reset by peer
```

To make the OPC connector and JSC work again, we had to change the TOPOLOGY PORTNUMBER to a different value (not equal to the SERVOPTS PORTNUMBER) and cycle the E2E Server task. This problem could occur if the JSC and E2E PROTOCOL functions were implemented in separate tasks (one task E2E only, one task JSC only) if the two PORTNUMBER values were set to the same value.

18.4 End-to-end scheduling Symphony switch and distribution (daily planning jobs)

In this section, we illustrate some problems that can either cause the Symphony file switch process to fail or cause the distribution of the Symphony file after a successful switch to fail. In addition to the examples listed here, some common problems that can cause a failure in the distribution of the Symphony file include:

- Incorrect procedures used to stop Tivoli Workload Scheduler on the fault-tolerant agents or domain managers
- Incorrect procedures used to stop the E2E Server task or its USS processes
- A file system that is full (either for USS or FTA)

These types of problems can cause corruption of the .msg files either on the USS or FTA. One indication of a corrupt .msg file (for example, tomaster.msg) is a file with a size larger than 48 bytes and that the size does not change for a long time.

We cover some common problems related to Symphony switching and distribution in the following sections.

18.4.1 EQQPT52E cannot switch to new Symphony file

The EQQPT52E message (text shown in Example 18-38), can be a difficult one for troubleshooting because there are several different possible causes.

Example 18-38 EQQPT52E message

```
EQQPT52E Cannot switch to the new symphony file:
run numbers of Symphony (x) and CP (y) aren't matching
```

The x and y in the message example would be replaced by the actual run number values. Sometimes the problem is resolved by running a SYMPHONY RENEW or CP REPLAN (or CP EXTEND) job. However, there are some other things to check if this does not correct the problem.

The EQQPT52E message can be caused if new FTA workstations are added through the Tivoli Workload Scheduler for z/OS dialog but the TOPOLOGY parameters are not updated with the new CPUREC information. In this case, adding the TOPOLOGY information and running a CP batch job should resolve the problem.

EQQPT52E can also occur if there are problems with the ID used to run the CP batch job or the E2E Server task. See 18.2, “Security issues with end-to-end feature” on page 616 for a discussion about how the security user IDs should be set up for end-to-end scheduling. One clue that a user ID problem is involved with the EQQPT52E message: if after the CP batch job completes, there is still a file in the WRKDIR whose name is Sym, plus the user ID that the CP batch job runs under. For example, if the CP EXTEND job runs under ID TWSRES9, the file in the WRKDIR would be named SymTWSRES9. If security were set up correctly, the SymTWSRES9 file would have been renamed to Symnew before the CP batch job ended.

If the cause of the EQQPT52E still cannot be determined, add the DIAGNOSE statements shown in Example 18-39 on page 630 to the parm file indicated.

Example 18-39 DIAGNOSE statements added

```
(1) CONTROLLER: DIAGNOSE NMMFLAGS('00003000')
(2) BATCH (CP EXTEND): DIAGNOSE PLANMGRFLAGS('00040000')
(3) SERVER : DIAGNOSE TPLGYFLAGS(X'181F0000')
```

Then collect this list of documentation for analysis:

- Controller and server EQQMLOGs
- Output of the CP EXTEND (EQQDNTOP) job
- EQQTWSIN and EQQTWSOU files
- USS stdlist/logs directory (or a tar backup of the entire WRKDIR)

18.4.2 CP batch job for end-to-end scheduling is run on wrong LPAR

The enqueues (ENQ) done for the E2E Server have a scope of SYSTEM, meaning they are only checked within the LPAR, not across LPARs if running in a sysplex environment. This means that if end-to-end scheduling is used, the CP batch jobs must be run on the same LPAR as the E2E Server task. This is true even if CHECKSUBSYS(YES) is coded in the BATCHOPT parameters. If a CP batch job for end-to-end scheduling is submitted to an LPAR other than the one on which the E2E Server is running, the job will hang indefinitely until it is cancelled. The relevant ENQs, which can be seen by issuing the command **D GRS,RES=(SYSZDRK,*)**, are shown in the following list. The actual subsystem name will replace *subn*:

- *subn*EQQTWSIE
- *subn*EQQTWSOE
- *subn*EQQTWSIF
- *subn*EQQTWSOF

18.4.3 No valid Symphony file exists

If a CP batch job ends with RC=12 and the following message is present in the EQQMLOG of the job, it indicates a situation where there is no Symphony file:

```
EQQ3091W OPC FAILED TO SYNCHRONIZE WITH E2E DISTRIBUTED ENVIRONMENT
```

If it were not intended to have an end-to-end scheduling environment or if it is critical to get the CP batch job to complete successfully to allow non-end-to-end scheduling processing to proceed, the following changes can be made:

- Comment out the TPLGYPRM keyword in the BATCHOPTS member.
- Comment out the TPLGYSRV parameter in OPCOPTS.
- Cycle (stop and then start) the Controller task.

After this, the CP batch job should run correctly, but end-to-end scheduling processing will be disabled.

18.4.4 DM and FTAs alternate between linked and unlinked

It is possible in end-to-end scheduling processing to create a situation that causes all of the DMs and FTAs to be linked for approximately 10 minutes, and then alternate to being unlinked for 10 minutes, and repeat this alternation between linked and unlinked. Possible reasons for this are:

- Non-existent FTAs are defined in the Tivoli Workload Scheduler for z/OS workstation (WS) database or in the TOPOLOGY parameters.
- A large portion of the distributed environment becomes unreachable by the primary domain Controller (PDM), for example, due to a network outage.

18.4.5 S0C4 abend in BATCHMAN at CHECKJOB+84

There are two APARs that correct problems resulting from an FTA operation incorrectly having more than 40 predecessors: PK14968 and PK17359.The following messages may appear in the E2E Server EQQMLOG.

Example 18-40 Messages in E2E Server EQQMLOG

```
EQQPH07E THE SERVER STARTER PROCESS ABENDED.
EQQZ045E OPC SUBTASK SERVER           ENDED UNEXPECTEDLY
```

From the dump, it is possible to find the occurrence token and operation number that is causing the problem.

Example 18-41 Dump output

```
Register 2 plus 4 bytes points to a data area.
At offset X'10' into this area will be an
operation number (example J001).
At offset X'3A' into this area will be the 16 byte
occurrence token value for the occurrence that has
caused the abend.
This 16-byte value can be input into the TWS 5.2
panel (EQQMOCCP) as "Occurrence token" to get more
information about the application and operation that
causes the abend.
```

When the operation has been found (from the occurrence token and operation number found in the dump), the following actions should be done with the E2E Server down:

1. Complete in the CP the operation found (the one with more than 40 dependencies).
2. Delete the Symphony file and Translator.chk file from the USS WRKDIR.
3. Restart the E2E Server.
4. Run a Symphony Renew job.

If no operations that remain in the Symphony cause the abend, then batchman will start. If another, different operation causes the same problem, find its occurrence token and operation number from the new dump as before and repeat the same circumvention steps.

Note: The Symphony Renew is necessary because the completion event will not reach the Symphony when batchman is down. The deletion of Symphony and Translator.chk is needed in order to get Translator to start. (If Translator is not started, the Symphony Renew cannot run.)

18.4.6 S0C1 abend in Daily Planning job with message EQQ2011W

Event Trigger Tracking (ETT) is used to add an end-to-end scheduling occurrence to the current plan. The command in the EQQSCLIB file was incorrect, so the set of messages in Example 18-42 was produced.

Example 18-42 Messages related with ETT

```
COMMAND SYNTAX ERROR: /WRAPPER.SH FURTHER STATEMENT PROCESSING STOPPED
BAD MEMBER xxxxxxxx CONTENTS IN EQQSCLIB ERROR DURING MCP UPDATE.
FAILING MODULE: EQQMCFLG 01/09/11 20.3 WRONG JOB DEFINITION FOR THE
FOLLOWING OCCURRENCE: ETT FAILED TO ADD F1FA0050 FOR TRIGGERING EVENT
eeeeeeee ERROR DETECTED BY MODIFY CURRENT PLAN INTERFACE
```

The ETT application has a successor dependency to an application in the current plan. When the format of the command was corrected, a replan was run to renew the symphony file. The replan abended with an abend S0C1and this message:

```
EQQ2011W A PREDECESSOR TO APPLICATION aaaaaaaa 040518  1800 WAS NOT
FOUND
```

An attempt to delete the dependency resulted in the message in Example 18-43.

Example 18-43 Messages related with ETT

```
EQQN043E UNSUCCESSFUL VSAM I/O REQUEST. DETAILED INFORMATION
FOLLOWS:
EQQN043I ACTIVE TASK IS GS EXECUTOR 01  . LOAD MODULE IS EQQMCTO
EQQMCTOP
EQQN043I I/O REQUEST IS FROM EQQMC3OC 03/05/28 14.17.24 UQ77237
    AT OFFSET +02D2
EQQN043I REQUESTED FUNCTION IS GU ON LOGICAL FILE CP , DDNAME
EQQCP1DS
EQQN043I VSAM RETURN CODE IS 8, REASON CODE IS 0016, KEY IS 03..
EQQN043I                          HEXADECIMAL ZONES    FF0030444
EQQN043I                          HEXADECIMAL DIGITS   030AE0000
```

This problem can be avoided if the PTF for PQ90560 is applied. However, if you get these symptoms and the PTF has not been applied yet, do not run a CP REPLAN or CP extend because the 0C1 abend will occur.

Instead, follow these steps:

1. Correct the member in the SCRIPTLIB associated to the predecessor application.
2. Add again the successor to the CP.
3. Run a REPLAN or CP EXTEND to prove that the CP corruption has been eliminated.
4. Apply the PTF.

18.4.7 EQQPT60E in E2E Server MLOG after a REPLAN

EQQPT36W UNABLE TO STOP CPU is seen in the USS E2EMERGE log after a CP REPLAN is run. Messages AWSDEC002E and AWSBCV012E are in the USS TWSMERGE log, and message EQQPT60E is seen in the E2E Server EQQMLOG. Other symptoms include some of the *.msg and/or pobox/*.msg files in the USS WRKDIR increasing in size and not returning to the original size (48).

This problem can be avoided by applying the PTF for PQ99358; however, the following procedure can be followed to bypass the problem:

1. Shut down the E2E Server task
2. Remove the affected .msg files from the USS WRKDIR
3. Start the E2E Server
4. Run a CP REPLAN

18.4.8 Symphony file not created but CP job ends with RC=04

The changes implemented in APAR PQ77014 changed the return code issued by a CP EXTEND or CP REPLAN job as in Example 18-44.

Example 18-44 Return codes

```
RC=8 -------> CP and Symphony not created.
RC=4 -------> CP created (Symphony yes or not)
```

If you have Warning or Error messages during the Symphony creation, the job will get RC=4. In this case the symphony could be created or not, depending on which of these messages is issued:

- EQQ3087I SYMNEW FILE HAS BEEN CREATED

  or:

- EQQ3088E THE SYMPHONY FILE HAS NOT BEEN CREATED)

For a Symphony renew job, the return codes are:

```
RC=8 -------> Symphony not created.
RC=4 -------> Symphony created.
```

If you have Warning or Error messages during the Symphony creation, you can get RC=4 or RC=8.

- If RC=4 then the EQQ3087I message will be issued:

  ```
  EQQ3087I SYMNEW FILE HAS BEEN CREATED
  ```

- If RC=8 then the EQQ3088E message will be showed:

  ```
  EQQ3088E THE SYMPHONY FILE HAS NOT BEEN CREATED.
  ```

An RC=04 from a daily planning job does *not* guarantee that the Symphony file has been created successfully. The EQQMLOG for the EQQDNTOP or EQQDRTOP jobstep will contain message EQQ3087I (success), or EQQ3088E (failure). The message must be checked.

To help automate this examination, the Tivoli Workload Scheduler for z/OS message library SEQQMSG0, member EQQ308, may be modified (as documented in the Miscellaneous Customization chapter of *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265) to cause message EQQ3088E to be issued as a WTO to the specified console as well as being written to the EQQMLOG of the jobstep.

18.4.9 CPEXTEND gets EQQ3091E and EQQ3088E messages

A current plan extend (EQQDNTOP) job is run using the end-to-end scheduling feature. The initial run fails with the messages in Example 18-45.

Example 18-45 Messages related with CPEXTEND

```
EQQ3106I WAITING FOR SCP
EQQ3091E OPC FAILED TO SYNCHRONIZE WITH THE END-TO-END
DISTRIBUTED ENVIRONMENT
EQQ3088E THE SYMPHONY FILE HAS NOT BEEN CREATED
```

However, if the CP EXTEND job is rerun a few minutes later, it runs correctly with the messages in Example 18-46.

Example 18-46 Messages related with CPEXTEND

```
EQQ3106I WAITING FOR SCP
EQQ3107I SCP IS READY: START JOBS ADDITION TO SYMPHONY FILE
```

This can be caused by the ENABLESWITCHFT parameter. The parameter ENABLESWITCHFT was set to (Y) in the TOPOLOGY parms. Changing this to ENABLESWITCHFT(N) corrected the problem. APAR PK08423 corrects two problems with the ENABLESWITCHFT parameter, one of which is the problem mentioned above.

If this problem occurs, try applying the fix for PK08423. If this does not help, try changing ENABLESWITCHFT to N. If this still does not correct the problem, report this to Tivoli Workload Scheduler for z/OS support (PMR with COMPID=5697WSZ01).

18.4.10 SEC6 abend in daily planning job

When a current plan batch job (CP EXTEND, CP REPLAN, or SYMPHONY RENEW) is run, an abend SEC6 with RSN 0F01C008 is received from module BPXJCPR. This is caused by a user ID that does not have an OMVS segment.

The 0F01C008 code indicates that the user does not have an OMVS segment. In order to run a CP batch job, a user ID must have an OMVS segment and must belong to the security group designated as eqqGID when the EQQPCS05 install job is run. In addition, the default group (DFLTGRP) for the user ID must *also* have a valid OMVS segment. Adding the OMVS segment and access to group eqqGID to the user ID enables the CP batch job to run without the SEC6 abend.

Note: APAR PK01415 adds serviceability improvements that would prevent the SEC6 abend. A message will be issued instead of the abend. Installing fix pack8 for USS (Tivoli Workload Scheduler for z/OS 8.2 PTFS UK06627 and UK06629) or higher will add the serviceability improvements in PK01415.

18.4.11 CP batch job starting before file formatting has completed

If the EQQTWSIN or EQQTWSOU file, or both, are newly allocated, they are formatted by the Controller task at startup time. If a CP batch job is started before the formatting of these files is complete, the job will fail with RC=12, and the following message will be seen in the EQQMLOG:

```
EQQ3120E END-TO-END TRANSLATOR SERVER PROCESS IS NOT AVAILABLE
```

This same message would appear if the E2E Server task were down when a CP batch job was started. After verifying that the E2E Server task was running, we checked the Controller EQQMLOG and found the messages shown in Example 18-47 on page 637.

Example 18-47 Excerpt from EQQMLOG

```
EQQW030I A DISK DATA SET WILL BE FORMATTED, DDNAME = EQQTWSIN
EQQW038I A DISK DATA SET HAS BEEN FORMATTED, DDNAME = EQQTWSIN
EQQG001I SUBTASK E2E RECEIVER HAS STARTED
EQQ3120E END-TO-END TRANSLATOR SERVER PROCESS IS NOT AVAILABLE
EQQW038I A DISK DATA SET HAS BEEN FORMATTED, DDNAME = EQQTWSOU
EQQG001I SUBTASK E2E SENDER HAS STARTED
EQQZ193I END-TO-END TRANSLATOR SERVER PROCESSS NOW IS AVAILABLE
```

The JESMSGLG of the Controller will also show D37 abends for the files that require formatting, as shown in Example 18-48.

Example 18-48 Excerpt from JESMSGLG

```
IEC031I D37-04,IFG0554P,TWSA,TWSA,EQQTWSIN,3595,SBOXB6,TWS.SC63.TWSIN
IEC031I D37-04,IFG0554P,TWSA,TWSA,EQQTWSOU,34CB,SBOXA8,TWS.SC63.TWSOU
```

If an RC=12 occurs in a CP batch job, check to make sure that the job was not submitted before the following message is seen in the controller EQQMLOG:

```
EQQG001I SUBTASK E2E SENDER HAS STARTED
```

If the batch job was submitted too early, restart the job and check for a successful completion code.

18.5 OMVS limit problems

Any of several OMVS limit problems can cause end-to-end scheduling processing to fail. The following console command (output in Example 18-49) can be used to get a current snapshot of key OMVS values, the high-water marks for these values, and the limit value as currently implemented:

```
D OMVS,L
```

Example 18-49 Output of the D OMVS,L command

```
BPXO051I 00.06.23 DISPLAY OMVS 412
 OMVS     000D ACTIVE          OMVS=(01,0A,00)
 SYSTEM WIDE LIMITS:         LIMMSG=ALL
                    CURRENT  HIGHWATER     SYSTEM
                      USAGE      USAGE      LIMIT
 MAXPROCSYS              38         45        200
 MAXUIDS                  7          8         50
 MAXPTYS                  1          2        256
 MAXMMAPAREA              0          0      40960
```

```
MAXSHAREPAGES          308        1934      131072
IPCMSGNIDS               0           0         500
IPCSEMNIDS               4           4         500
IPCSHMNIDS               0           0         500
IPCSHMSPAGES             0           0      262144
IPCMSGQBYTES           ---           0      262144
IPCMSGQMNUM            ---           0       10000
IPCSHMMPAGES           ---           0         256
SHRLIBRGNSIZE            0           0    67108864
```

The next sections illustrate the effect of having the values for MAXFILEPROC, MAXPROCSYS, and MAXUIDS set too low.

18.5.1 MAXFILEPROC value set too low

To test the effect of having MAXFILEPROC set too low, we issued this command:

```
SETOMVS MAXFILEPROC=5
```

Note that the normal value on the system used for the test was 2000, which is set in the BPXPRMxx member in the system PARMLIB. The current value of MAXFILEPROC is also displayed if the following command is issued:

```
D OMVS,O
```

With MAXFILEPROC=5 in effect, the E2E Server was started, and the messages shown in Example 18-50 were seen in the EQQMLOG.

Example 18-50 Excerpt from EQQMLOG

```
EQQPH33I THE END-TO-END PROCESSES HAVE BEEN STARTED
EQQZ024I Initializing wait parameters
EQQPH07E THE SERVER STARTER PROCESS ABENDED.
EQQPH07I THE STARTER PROCESS WAS CREATED TO PROCESS E2E REQUESTS
```

The TWSMERGE log in USS stdlist had the following message for the domain manager (HR82 in this example) and the FTAs:

```
MAILMAN:+ AWSBCV027I Unlinking from HR82
```

After correcting the MAXFILEPROC value with the following command and cycling the E2E Server, the problem was corrected:

```
SETOMVS MAXFILEPROC=2000
```

The conclusion is that if message EQQPH07E is seen in the server EQQMLOG, the value of MAXFILEPROC should be checked and increased if needed.

18.5.2 MAXPROCSYS value set too low

We tested a small value for MAXPROCSYS in a similar manner to the way MAXFILEPROC was tested in the previous section. The current value for MAXPROCSYS can be seen with either the **D OMVS,L** command or the **D OMVS,O** command. We issued the following command to set a small value for MAXPROCSYS:

```
setomvs maxprocsys=5
```

After this, the E2E Server task was started. The messages shown in Example 18-51 appeared in the server EQQMLOG.

Example 18-51 Excerpt from EQQMLOG

```
EQQZ005I OPC SUBTASK SERVER          IS BEING STARTED
EQQPH09I THE SERVER IS USING THE TCP/IP PROTOCOL
EQQPH18E COMMUNICATION FAILED,
EQQPH18I THE SOCKET     SOCKET CALL FAILED WITH ERROR CODE    156
EQQPH08I TCP/IP IS EITHER INACTIVE OR NOT READY
EQQPH08I CHECK THAT TCP/IP IS AVAILABLE
EQQPH00I SERVER TASK HAS STARTED
EQQPH18E COMMUNICATION FAILED,
EQQPH18I THE SOCKET     SOCKET CALL FAILED WITH ERROR CODE    156
EQQPH08I TCP/IP IS EITHER INACTIVE OR NOT READY
EQQPH08I CHECK THAT TCP/IP IS AVAILABLE
EQQPH35E CANNOT START STARTER PROCESS:
   EQQPH35I BPX1ATX FAILED WITH RC=0156, RSN=0B0F0028
```

We used the TSO BPXMTEXT command to check the meaning of the reason code shown in the EQQPH35I message (0B0F0028):

```
TSO BPXMTEXT 0B0F0028
```

The information shown in Example 18-52 was displayed by BPXMTEXT.

Example 18-52 Output of the TSO BPXMTEXT 0B0F0028 command

```
JRMaxProc: The maximum number of processes was exceeded
Action: Retry after some processes have ended, or change the maximum
number of processes allowed.
```

To correct the problem, MAXPROCSYS was reset to its normal value with this command, and the E2E Server was stopped and started again:

```
setomvs maxprocsys=200
```

18.5.3 MAXUIDS value set too low

We tested a low value for MAXUIDS in a manner similar to the tests for MAXFILEPROC and MAXPROCSYS in the previous sections. Again, the value for MAXUIDS can be displayed using either the **D OMVS,O** command or the **D OMVS,L** command. The following command was issued to set the MAXUIDS value:

```
SETOMVS MAXUIDS=2
```

The following console message appeared immediately, but we ignored it and attempted to start the E2E Server anyway:

```
*BPXI039I SYSTEM LIMIT MAXUIDS HAS REACHED 300% OF ITS CURRENT CAPACITY
```

The messages shown in Example 18-53 were in the server EQQMLOG.

Example 18-53 Excerpt from EQQMLOG

```
EQQPH09I THE SERVER IS USING THE TCP/IP PROTOCOL
EQQPH18E COMMUNICATION FAILED,
EQQPH18I THE SOCKET     SOCKET CALL FAILED WITH ERROR CODE    156
EQQPH08I TCP/IP IS EITHER INACTIVE OR NOT READY
EQQPH08I CHECK THAT TCP/IP IS AVAILABLE
EQQPH00I SERVER TASK HAS STARTED
EQQPH18E COMMUNICATION FAILED,
EQQPH18I THE SOCKET      SOCKET CALL FAILED WITH ERROR CODE     156
EQQPH08I TCP/IP IS EITHER INACTIVE OR NOT READY
EQQPH08I CHECK THAT TCP/IP IS AVAILABLE
EQQPH35E CANNOT START STARTER PROCESS:
EQQPH35I BPX1ATX FAILED WITH RC=0156, RSN=130C0013
```

These messages are similar to those issued when MAXPROCSYS was set too low, except for the reason code in the EQQPH35I message, which is 130C0013. We issued the following command to check the meaning of this RSN code:

```
TSO BPXMTEXT 130C0013
```

The BPXMTEXT display included the output shown in Example 18-54.

Example 18-54 Output of the TSO BPXMTEXT 130C0013 command

```
JRMaxUIDs: The maximum number of OpenMVS user IDs is exceeded
Action: Ask the system programmer or system administrator to increase
the MAXUIDS parameter of the BPXPRMxx parmlib member.
```

This was corrected by issuing this command and cycling the E2E Server task:

```
setomvs maxuids=50
```

18.6 Problems with jobs running on FTAs

We provide some examples of problems related to jobs running on fault-tolerant agents (FTAs) along with the fix or workaround.

18.6.1 Jobs on AS/400 LFTA stuck Waiting for Submission

After applying the fix for APAR PQ85035, any schedule with external dependencies for an AS400 LFTA are going into “stuck” status. The error in Example 18-55 is seen in the PDM MAESTRO log and the LFTA log.

Example 18-55 AS400 LFTA messages

```
BATCHMAN:09:15/Received Hi: AS4B
BATCHMAN:09:15/************************************************
BATCHMAN:09:15/* AWS22010008E Invalid mailbox record received:
Hi
BATCHMAN:09:15/************************************************
```

Stopping on the AS400, renaming the Intercom.msg file and then restarting TWS did not help.

The solution is to apply the fix for PK02652.

18.6.2 Backslash “\” may be treated as continuation character

When coding a centralized script to reside in the Controller EQQJBLIB for an FTW operation, if the last character of any line is a backslash (EBCDIC x'E0', ASCII x'5C'), that character is treated by the mainframe code as a continuation character. That is, everything prior to it on that line is concatenated to the following line, and the “\” character is deleted.

When creating scripts for a Windows FTA, this may cause problems, because a Windows PATH may end with a backslash character. This problem appears after applying the fixing PTF for APAR PQ93878, which corrects a problem with variable substitution in centralized scripts.

If any line in a centralized script ends with a backslash that you do not want to be treated as a continuation character, place an ampersand (&) immediately after the backslash. Thus the backslash will not be the last character on the line and the mainframe code will not treat it as a continuation. When the script is executed on a Windows platform, the trailing ampersand signifies a compound command, and because nothing follows it, it has the effect of inserting a blank line into the script (that is, it has no effect at all on the execution of the script).

For instance, assume the script in Example 18-56 is executed.

Example 18-56 Script executed

```
@ECHO OFF
set PTH=C:\DIR1\DIR2\DIR3\&
set PGM=ANYTHING.EXE
```

Then the following command has the result shown, which is the same as if there were no ampersand at the end of the first SET command.

Example 18-57 Output of the script

```
echo %PTH%%PGM%
C:\DIR1\DIR2\DIR3\ANYTHING.EXE
```

18.6.3 FTA joblogs cannot be retrieved (EQQM931W message)

In an attempt to retrieve a joblog for an FTA job from the TSO dialog, message EQQM931W is issued. Checking the EQQMLOG of the E2E Server task shows message EQQPT36W that refers to another message, AWSBIN080E. APAR PK09036 refers to a similar problem, but this has already been applied. The cause of this problem may be the FIREWALL parameter in CPUREC.

Whenever an FTA joblog cannot be retrieved, it is possible that FIREWALL(YES) has to be specified in the CPUREC for that FTA. Add the FIREWALL(YES) parameter, then run a CP REPLAN or CP EXTEND job. If the joblog retrieval problem is *not* eliminated, make sure that the PTF for PK09036 has been properly applied (Tivoli Workload Scheduler for z/OS 8.2 PTF UK06043).

To verify that UK06043 is applied, browse the SEQQLMD0 library used by the DP batch jobs (CP REPLAN, CP EXTEND, SYMPHONY RENEW), either in the STEPLIB of the JCL or in the LINK LIST (LNKLST). Select load module EQQDNTOP and do a FIND for EQQDPSY2, then do a REPEAT FIND (F5). You see a display of EQQDPSY2 and the LAST PTF that updated it, for example:

```
EQQDPSY2 05/11/18 16.39.40 UK09283
```

The PTF number could be UK06043, UK07068, UK09283, or a later PTF. If the PTF number is *earlier* than UK06043, it indicates that the PTF is either not applied at all, or that it was applied, but load module EQQDNTOP was not copied into the correct library (or libraries) afterward.

18.6.4 FTA job run under a non-existent user ID

A job running on an FTA in an end-to-end scheduling network fails (it goes to the FAIL state). The error AWS21000009 can occur in the end-to-end scheduling environment if the user defined in the JOBUSR parameter of the JOBREC does not exist on the FTA where the job runs.

The error in Example 18-58 appears in the MAESTRO log.

Example 18-58 Errors in the MAESTRO log

```
JOBMAN:07:16/++++++++++++++++++++++++++++++++++++++++++++++++++
JOBMAN:07:16/+Error launching
JOBMAN:07:16/+OPCMASTER#B9E4BEF18672AD86.J_015_LIMITTEST1:
JOBMAN:07:16/+AWS21000009 Unable to retrieve the passwd structure.
JOBMAN:07:16/+Errno =The process does not exist..
JOBMAN:07:16/++++++++++++++++++++++++++++++++++++++++++++++++++
```

Change the JOBREC corresponding to this job (in the script library on the mainframe) so that the JOBUSR parameter is set to the correct FTA user. Then renew the Symphony file from the Tivoli Workload Scheduler for z/OS TSO dialog panels (=3.5). When the new Symphony has been created and sent to the FTA, rerun the failed job by changing its status in the Tivoli Workload Scheduler for z/OS dialog to R (Ready).

18.6.5 FTA job runs later than expected

An FTA job that is time dependent but has no predecessor dependencies starts later than expected. No indication can be found of this in the FTA or USS TWSMERGE logs. This type of problem can be caused by a hardware reserve on the mainframe.

A hardware reserve on a DASD volume containing OMVS files, or the EQQTWSIN and EQQTWSOU files allocated to the E2E Server task, can cause a delay in end-to-end scheduling jobs being started. Check the MVS SYSLOG and look for messages similar to Example 18-59.

Example 18-59 MVS SYSLOG

```
IEF196I IOS071I 40BB,**,OMVS, START PENDING
IOS431I DEVICE 40BB RESERVED TO CPU=0115A22064,LPAR ID=1 580
SYSTEM=PRDB
ISG343I 00.10.36 GRS STATUS 014 584
DEVICE:40BB VOLUME:P$HU06 RESERVED BY SYSTEM PRDB
```

Any messages indicating START PENDING or DEVICE RESERVED for either OMVS or the E2E Server task are likely to cause a delay in end-to-end scheduling processing. To avoid this type of problem, do not place files used by OMVS or the E2E Server task on DASD volumes that are subject to hardware reserves.

18.6.6 FTA jobs do not run (EQQE053E message in Controller MLOG)

The CP EXTEND or CP REPLAN or SYMPHONY RENEW jobs run clean. The SYMPHONY is successfully distributed, FTAs link, and ad hoc work runs correctly. However, scheduled FTA jobs do not run and the Controller EQQMLOG contains many occurrences of messages EQQE053E and EQQE053I. JOB TOKENS for jobs in the CURRENT PLAN do not match same jobs in the SYMPHONY file, and the tokens in the SYMPHONY never change even when a new SYMPHONY is created. The cause is incorrect EQQSCPDS file allocation.

The EQQSCPDS DD statement in the Controller started task points to a different data set than the EQQSCPDS DD statement specified in the CP BATCH (EXTEND, RENEW, SYMPHONY RENEW) JCL. These DD statements must all reference the same DATASET. Correct the EQQSCPDS DD statements in the CP batch JCL and the Controller proc so they all point at the same data set, then run a CP EXTEND, CP REPLAN, or SYMPHONY RENEW batch job.

18.6.7 Jobs run at the wrong time

Several APARs have dealt with problems that cause FTA jobs to run at the wrong time (for example, PQ74098 and PQ74464). However, if end-to-end scheduling maintenance is up to date, and FTA jobs are running at the wrong time, the following items should be checked:

- Review APAR PQ90090 regarding correct setup of the CPUTZ() value in the CPUREC.
- The GMT (hardware) clock on the z/OS system where the Controller runs must be set to GMT.
- The LOCAL (software) clock on the z/OS system where the Controller runs must be set to LOCAL time.
- The TIME/DATE and TIMEZONE must be set correctly on the machine where the FTA runs.
- In the GLOBALOPTS of the FTA, “Timezone enable” must be set to YES.
- Use the information in INFO APAR II13900 to download the latest list of recommended end-to-end scheduling maintenance and make sure that it is applied.

18.7 OPC Connector troubleshooting

This troubleshooting focuses on the communication aspect of the OPC Connector. As seen in 16.2, “Activating support for the Job Scheduling Console” on page 483, two pieces are required to be installed and configured for the JSC connection to work in the end-to-end scheduling environment:

- The first is that the OPC Connector has been installed.
- The second is that there is a Tivoli Workload Scheduler for z/OS Server running. This Tivoli Workload Scheduler for z/OS Server has been configured using the OPCOPTS, SERVOPTS, and TOPOLOGY statements.

OPC Connector installation

For the connector to work, the Tivoli Framework, the Job Scheduling Services, and the OPC Connector should be installed. The command to check what is installed is **`wlsinst -ah`**. Remember to source the Tivoli Framework environment with **`setup_env.sh`** (UNIX) or **`setup_env.exe`** (Windows).

The entries in Example 18-60 should be seen in the results of the **`wlsinst`** command if the JSC 1.3 OPC Connector has the most recent fix pack (as of April 2006). The Job Scheduling Console V1.4 is distributed with Tivoli Workload Scheduler V8.2.1; it also included the JSS V1.2 and Framework V4.1.1, so the difference will be seen in the connector version and lack of fix pack (upgrade).

Example 18-60 Entries seen in the results of the wlsinst command

```
Tivoli Management Framework 4.1.1
Tivoli Workload Scheduler for z/OS connector V 1.3
Tivoli Job Scheduling Services V1.2
Tivoli Workload Scheduler for z/OS connector upgrade to v1.3 fixpak 9.
```

EQQMA05E and EQQMA17E: fix packs and networking issues

The most common problem after both the mainframe and distributed parts have been installed and configured is this message showing up during the creation of the instance using the **`wopcconn`** command. If you are seeing the EQQMA05E and EQQMA17E, you should start with these solutions, then move to the EQQMA11E solutions if the connection is still failing.

Example 18-61 EQQMA05E and EQQMA17E messages

```
EQQMA05E  The OPC Connector cannot communicate with the OPC Server and
retrieve version information.
EQQMA17E TCP/IP socket I/O error...
```

This shows that the connector cannot get the version of the TWS from the mainframe. Communication has not been established, and the problem will be found in either the configuration on the mainframe or the distributed. Check the in this order:

1. If you are running Tivoli Workload Scheduler V8.2 and the OPC Connector V1.3, put on the latest fix pack for the OPC Connector. The fix pack can be obtained at:

   ftp://ftp.software.ibm.com/software/tivoli_support/patches/patches_1.3.0

   The most recent fix pack as of April 2006 is 1.3.0-TIV-TWSJSC-FP0009. Tivoli Workload Scheduler V8.2.1 with JSC V1.4 does not need a fix pack for this issue.

2. Check that your configuration has both the OPC Connector and the JSC Server pointing to the same IP address. The IP address or host name entered with the **wopcconn** command has to point to the mainframe where Tivoli Workload Scheduler for z/OS resides. The same TCP/IP address should be used in the JSCHOSTNAME() variable in the SERVOPTS for the JSC server on the Tivoli Workload Scheduler for z/OS mainframe. Both IP addresses should point to the same mainframe IP stack. This should work even with VIPA as long as both dotted-decimal addresses match.

   Ports should also be verified, including verifying that the ports on both ends are properly assigned by the operating systems and that firewalls are allowing the communication through.

EQQMA11E, EQQPH26E: USERID mapping

When configuring the OPC Connector, EQQMA05E often is accompanied by the EQQMA11E. Often, the fixes above for the EQQMA05E will solve other issues. Error EQQMA11E is often seen when the user IDs are not resolving. Here is an outline to check user ID resolution with the OPC Connector.

The OPC Connector requires three IDs to connect to each other for authorization. The user ID that the customer uses to log on to the JSC Console on their local system must be authorized to a user ID for the Tivoli Framework on the OPC Connector system. The Tivoli Framework user ID must be associated with an RACF (or alternative mainframe security) user ID on the mainframe.

An EQQMA11E error may appear as shown in Example 18-62.

Example 18-62 EQQMA11E error

```
EQQMA11E Cannot allocate connection: Root_sharon-region@sharon-region -
TCPIP - 123.123.123.5:7100 ( 1 )
```

Note the user ID in this example is `Root_sharon-region@sharon-region` and is a Tivoli Framework user ID. The Tivoli Framework manuals explain how to set up and maintain the relationship between system user IDs and Tivoli Framework user IDs. When we log in to the JSC, we use an ID that is locally defined to start the JSC. That user ID is mapped to Framework user ID Root_sharon-region@sharon-region.

The relationship between the Tivoli Framework user ID and the RACF user ID can be configured in one of two ways:

- The first method has a limitation that= requires stopping and restarting the JSC Server on the mainframe to add new users. In short, set the USERMAP parameter on the SERVOPTS statement to point to a member in the EQQPARM DD statement. Entries in this member will look like the following examples.

Example 18-63 Set the USERMAP parameter to point to a member in the EQQPARM DD

```
USER 'Root_twsztest-region@twsztest-region'       RACFUSER(ATKN)
USER 'Cy@twsztest-region'                         RACFUSER(ATKN)
USER 'E2E@horrible.tivlab.austin.ibm.com-region'  RACFUSER(USR191)
```

  Be very careful copying the Tivoli Framework user ID into the EQQPARM member. We recommend copying and pasting it into the mainframe editor.

- The second method requires direct RACF administration work by an RACF administrator. By using the RACF group TMEADMIN, users can be added to and removed from the group as needed without stopping and restarting the JSC Server.

  Example 18-64 shows a sample of the RACF commands needed to set up the TMEADMIN class.

Example 18-64 RACF commands

```
SETROPTS CLASSACT(TMEADMIN)
RDEFINE TMEADMIN Root_twsztest-region@twsztest-region  -
APPLDATA('WHEELER')
SETROPTS RACLIST(TMEADMIN) REFRESH

The Tivoli Framework userid is Root_twsztest-region@twsztest-region.
The RACF userid is WHEELER
The RACF group is TMEADMIN
```

Tip: We have one final note on configuring the relationship between the Tivoli Framework user ID and the RACF user ID. We are often asked: If you use both a TMEADMIN group and the USERMAP() data set, which one will be used by Tivoli Workload Scheduler for z/OS?

If a USERMAP() keyword is coded, the TMEADMIN RACF class is totally ignored. Only those users defined in the USERMAP() will have access.

18.8 SMP/E maintenance issues

We present some end-to-end scheduling problems that can be introduced through the application of SMP/E maintenance and also some recommendations for applying maintenance.

18.8.1 Message CCGLG01E issued repeatedly; WRKDIR may be full

Several messages with code CCGLG01E can be found in the stdout file of the WRKDIR/stdlist directory under USS. The number of messages logged can be very high, then causing a large amount of useless space occupancy.

The message is logged by the CClog (the logging engine used by Tivoli Workload Scheduler for z/OS) when a logger handler is not defined in the TWSCCLog.properties file.

APAR PK11341 has introduced the capability to send a customizable set of end-to-end scheduling messages to the Server EQQMLOG and to the MVS Console. The APAR has created a new version of the TWSCCLog.properties file located in the BINDIR/config directory on USS. Customers have been instructed (by one of the ACTION HOLDS for PTF UK09596) to manually copy the new version of the TWSCCLog.properties file from BINDIR/config to WRKDIR.

If the old version of the TWSCCLog.properties file is used when APAR PK11341 is applied, message CCGLG01E will be issued by the CClog. The message will be logged in the WRKDIR/stdlist/stdout file and in several files under the WRKDIR/stdlist/<date> directories, where <date> is the current date.

18.8.2 Messages beginning EQQPH* or EQQPT* missing from MLOG

After applying maintenance to FMID HWSZ203 (specifically, PTF UK09601 for APAR PK11341), all messages originating from the end-to-end scheduling code in USS are missing from the E2E SERVER EQQMLOG. These are messages with an EQQPH or EQQPT message ID.

This problem is caused when the HOLD ACTION item for PTF UK09601 is not performed. After applying PTF UK09601, the updated default TWSCCLog.properties file in the BINDIR/config directory must be manually copied into the WRKDIR, replacing the old version of the file that is already there. APAR PK11341 changes the USS E2E code so that *all messages* issued to the WRKDIR/stdlist directory can be forwarded to the E2E Server EQQMLOG, or even to the SYSLOG. Which messages are forwarded, and where, is controlled by the TWSCCLog.properties file. Unless the updated sample of this file is implemented, no messages at all from the USS are sent to the SERVER EQQMLOG or to the SYSLOG.

Also, after applying these PTFs, nothing is ever written to the EQQMLE2E DD, and this DD statement can be removed from the E2E SERVER JCL.

Applying the PK11341 PTFs creates a new member in the SEQQMISC library, EQQPDFML. This file contains all of the documentation updates related to this functional change. To view these updates, download member EQQPDFML to a workstation in binary mode, name it `EQQPDFML.pdf`, and open it with Adobe Acrobat Reader.

18.8.3 S0C4 in E2E Server after applying USS fix pack8

The PTFs for fix pack8 (UK06627 and UK06629 for Tivoli Workload Scheduler for z/OS 8.2) were installed and the listed ACTION HOLDS were performed. After restarting the Controller and E2E Server, we saw the S0C4 abend in Example 18-65. (`PSW` points to CGETDMP+C8.)

Example 18-65 S0C4 abend messages

```
PSW=078D1400 916CF900
General purpose register values
0-3 00000011 00000000 00000000 0E5AF582
4-7 00000011 00000042 0E893190 0E8931B0
8-11 0E85183C 0E848998 0E893190 0E8931B0
12-15 0E8CBBD8 0E8F3098 0E8E73D0 00000154
```

The E2E Server was restarted and processing continued normally (no additional abends). The cause of the abend was determined to be that PK10713 requires EQQTWSOU to be empty (no in-flight records).

Analysis of the dump showed that the record being processed was written *before* the new maintenance level was applied (that is, the record was in-flight). To avoid the abend, before installing the fix pack8 maintenance, a controlled shutdown of the end-to-end scheduling environment should be done to ensure that there are no unprocessed records in the EQQTWSIN or EQQTWSOU files. This will mean

ACTION HOLD for any future PTFs where there is a change in the end-to-end scheduling record definitions that causes an incompatibility between a record written at the *old* maintenance level and a record written at the *new* maintenance level.

See 18.8.4, “Recommended method for applying maintenance” on page 650 for a description of how to perform the controlled shutdown.

18.8.4 Recommended method for applying maintenance

When SMP/E maintenance is applied directly to an end-to-end scheduling environment or copied from one end-to-end scheduling environment to another (for example, from a test to a production end-to-end scheduling system) you should always ensure that all affected items are properly copied over and that all JCL points to the correct version of any modified libraries. This can include:

- Load modules (SEQQLMD0)
- The USS BINDIR
- Changes that must be applied to the WRKDIR
- JCL changes (PROCs, batch jobs for daily planning
- Skeleton JCL used to create daily planning jobs from the dialog
- Parameter changes that are needed (EQQPARM member)

However, in some cases the end-to-end scheduling environment must be quiesced properly (that is, shut down in a controlled manner) to avoid problems on the subsequent restart. The following procedure must be used in these cases. (It can be used any time maintenance is being migrated to an end-to-end scheduling environment to avoid any problems that may not have been documented in the ACTION HOLDS for the PTFs.)

This is the procedure to ensure that the EQQTWSIN and EQQTWSOU files are EMPTY before starting the Controller and E2E Server tasks to install a new level of maintenance:

1. Deactivate FTA job submission (Tivoli Workload Scheduler for z/OS dialog option 9.1, then FY option).
2. Wait for EQQTWSOU to empty (see EMPTY procedure below).
3. Stop the E2E Server task.
4. Wait for EQQTWSIN to empty (see EMPTY procedure below).
5. Stop the Controller task.
6. Restart Controller task (which will restart the E2E Server task).

EMPTY procedure (check whether EQQTWSIN or EQQTWSOU file is EMPTY)

The first record in the EQQTWSIN and EQQTWSOU files is a HEADER record, with the following layout:

```
bytes 1-4 = HEAD
bytes 9-12 = write cycle number
bytes 13-16 = read cycle number
bytes 25-28 = next record to write
bytes 29-32 = next record to read
```

Turn HEX on in TSO BROWSE or EDIT to see these fields. For example:

```
00000000011111111112222222222333
12345678901234567890123456789012 (column indicator)
HEAD...................á...ø...ø....AO.....................
CCCC00000000000000020004000700070000CD0000000000000000000000
85140000000400040OC0000500100010000016000000000000000000000
```

If `next record to write` is equal to `next record to read`, then the data set is empty if the write cycle number is equal to the read cycle number (scenario a) or full if the write cycle number is different from the read cycle number (scenario b).

In the example above, next record to write is X'00000170' and next record to read is X'00000170'.The write cycle number is X'00000004' and the read cycle number is also X'00000004'. Therefore, according to the rules above, the data set is empty.

18.8.5 Message AWSBCV001E at E2E Server shutdown

Sometimes when the Tivoli Workload Scheduler for z/OS E2E Server is stopped. message AWSBCV001E can be sent by mailman and logged in the TWSMERGE log file in the USS stdlist. After APAR PK01415 is installed, the batchman process has the same GID (group ID) of the netman process.

At server shut down, netman sends a SIGKILL to all processes having the same GID as netman, thus killing batchman also. When mailman tries to stop batchman, it may be that batchman has already died. If so, the following message is sent:

```
MAILMAN:+ AWSBCV001E Error Batchman abended, Status: AWSDCJ201I
Terminated normally
```

If this message is received at E2E Server shut down, it can be ignored.

18.9 Other end-to-end scheduling problems

In the following sections, we present some problems that do not fit into any of the other categories.

18.9.1 Delay in Symphony current plan (SCP) processing

If an unusual delay is noticed between the time stamp of the two messages shown in Example 18-66 in the EQQMLOG of the CP batch job, a problem could be indicated.

Example 18-66 Excerpt from EQQMLOG

```
EQQ3106I WAITING FOR SCP
EQQ3107I SCP IS READY: START JOBS ADDITION TO SYMPHONY FILE
```

This might be a normal delay, especially if the network of domain managers and FTAs is large. After the EQQ3106I message is issued, the server issues STOP commands to OPCMASTER (batchman in the server) and all FT workstations. Then, the server waits until all the stopped processes and FTAs return an acknowledgment and signals to the normal mode manager (NMM) that the process has completed. The NMM applies the JT and finally copies the updated new current plan into CP1, CP2, and SCP. At this point, the message EQQ3107I is issued.

However, it is also possible that the problem is not related just to the number of FTAs that need to be stopped. There could be a problem within the TCP/IP network or the communication between Tivoli Workload Scheduler for z/OS and TCP/IP. APAR PQ92466 deals with such a problem.

18.9.2 E2E Server started before TCP/IP initialized

Prior to the fix for APAR PQ90369, if the E2E Server task was started before the TCP/IP task was fully initialized, the messages shown in Example 18-67 appeared in the server EQQMLOG.

Example 18-67 Excerpt from EQQMLOG

```
EQQPH09I THE SERVER IS USING THE TCP/IP PROTOCOL
EQQPH18E COMMUNICATION FAILED,
EQQPH18I THE SOCKET      SOCKET CALL FAILED WITH ERROR CODE    1036
EQQPH08I TCP/IP IS EITHER INACTIVE OR NOT READY
EQQPH08I CHECK THAT TCP/IP IS AVAILABLE
EQQPT35E Unable to Open Symphony and/or events files, Reason:
          AWSBDY102E Attempt to use NULL or uninitial
```

```
EQQPT40I Output Translator thread is shutting down
EQQPT09E The Netman process (pid=27) ended abnormally
```

After TCP/IP was initialized, the following message was written to the server EQMLOG. However, the server task did not recover correctly.

```
EQQPH28I THE TCP/IP STACK IS AVAILABLE
```

The server task had to be cycled in order to connect to TCP/IP correctly.

With the PQ90369 fix applied, the messages shown in Example 18-68 occur in the server EQQMLOG if the server is started prior to TCP/IP being initialized.

Example 18-68 Excerpt from EQQMLOG

```
EQQPH09I THE SERVER IS USING THE TCP/IP PROTOCOL
EQQPH18E COMMUNICATION FAILED,
EQQPH18I THE SOCKET      SOCKET CALL FAILED WITH ERROR CODE    1036
EQQPH08I TCP/IP IS EITHER INACTIVE OR NOT READY
EQQPH08I CHECK THAT TCP/IP IS AVAILABLE
EQQPH00I SERVER TASK HAS STARTED
EQQPH18E COMMUNICATION FAILED,
EQQPH18I THE SOCKET      SOCKET CALL FAILED WITH ERROR CODE    1036
EQQPH08I TCP/IP IS EITHER INACTIVE OR NOT READY
EQQPH08I CHECK THAT TCP/IP IS AVAILABLE
```

However, after TCP/IP is initialized, the messages in Example 18-69 are seen in the server EQQMLOG, and the server processes normally (no need to cycle).

Example 18-69 Excerpt from EQQMLOG

```
EQQPH28I THE TCP/IP STACK IS AVAILABLE
EQQPH37I SERVER CAN RECEIVE JSC REQUESTS
```

In addition, a new message has been added to indicate that the server will retry the TCP/IP connection every 60 seconds (Example 18-70).

Example 18-70 Excerpt from EQQMLOG

```
EQQPT14W TCPIP stack is down. A connection retry will be attempted in
60 seconds.
```

18.9.3 CPUTZ defaults to UTC due to invalid setting

If the CPUTZ value in CPUREC is set to a value for which there is no file defined in the bindir/zoneinfo/ directory, the action taken is to default the time zone for

that FTA to UTC (GMT+0). To illustrate this, we set up a CPUREC definition that had CPUTZ incorrectly specified as:

```
CPUTZ(/America/Chicago)
```

The correct definition would *not* have the leading slash:

```
CPUTZ(America/Chicago)
```

When a CP EXTEND was run with the incorrect CPUTZ value, the messages shown in Example 18-71 were seen in the CP batch job EQQMLOG.

Example 18-71 Excerpt from EQQMLOG

```
EQQ3105I A NEW CURRENT PLAN (NCP) HAS BEEN CREATED
EQQ3103E EQQputcp : UNABLE TO LOAD TIMEZONE INFORMATION FOR
/America/Chicago
EQQ3103I TIMEZONE UTC WILL BE USED FOR CPU HR82
EQQ3099E THE REASON OF THE PREVIOUS ERROR IS:
EQQ3099I EDC5129I No such file or directory.
EQQ3106I WAITING FOR SCP
```

The CP EXTEND runs to completion despite the error. However, any jobs set to run on the affected FTA would start at the wrong time. Therefore, if jobs are running at the wrong time on some FTAs, check the EQQMLOG for the last CP batch job that was run, and look for message EQQ3103E.

18.9.4 Domain manager file system full

We decided to test the effect of creating a Symphony file that is too large to fit into the file system used by the domain manager. The application database was set up so that the number of occurrences that would be added to the Symphony file would produce a Symphony file of about 450 MB. When the CP EXTEND was run, there were no messages in the E2E Server EQQMLOG or the Controller EQQMLOG concerning the file system problem. However, in the TWSMERGE log of the domain manager, message AWSBCU010E appeared several times. The following message was seen repeatedly in the USS TWSMERGE log:

```
AWSBCV082I Cpu AXTS, Message: AWSDEB003I Writing socket: EDC5140I
Broken pipe.
```

If repeated AWSBCV082I messages are seen in the USS TWSMERGE log, check the file system for the FTA mentioned in the message and take corrective action if the file system is full or nearly full.

18.9.5 EQQW086E in Controller EQQMLOG

If the message shown in Example 18-72 is seen in the Controller EQQMLOG, there might be a PTF ACTION HOLD that was not done.

Example 18-72 Excerpt from EQQMLOG

```
EQQW086E ERROR IN RECEIVER SUBTASK OF END-TO-END ENABLER TASK, IN
EQQTWDQI MODULE EQQW086I DURING A EVENT HANDLINGUN FUNCTION (ERROR CODE
= 16).
```

Check whether the PTF for APAR PQ77970 was applied. The ACTION HOLD for this PTF was missing an item, and this information was added in INFO APAR II13859. To avoid the EQQW086E message, use the following procedure:

1. Make sure that there are no JCL downloads and joblog retrievals pending.
2. Stop the Tivoli Workload Scheduler E2E Server.
3. Delete the <workdir>/Translator.wjl file.
4. Restart the server, and the Translator.wjl file will be reallocated automatically.

The next time the Controller is started, the EQQW086E message will not appear in the EQQMLOG.

18.9.6 S0C4 abend in E2E Server task DO_CATREAD routine

An abend S0C4 can occur in the E2E Server with the messages in Example 18-73 also being seen.

Example 18-73 Messages related with S0C4 abend

```
EQQ3120E END-TO-END TRANSLATOR SERVER PROCESS IS NOT AVAILABLE
IEW4000I FETCH FOR MODULE IDIXDCAP FROM DDNAME -LNKLST- FAILED
BECAUSE INSUFFICIENT STORAGE WAS AVAILABLE.
```

The dump shows that the abend occurs in the DO_CATREAD routine of BATCHMAN. The solution is to apply the PTF for PK1743.2

18.9.7 Abend S106-0C, S80A, and S878-10 in E2E or JSC Server

After a server with PROTOCOL(JSC) or PROTOCOL(E2E) has been running for some time, it abends with S106-0C, S80A, and S878-10. This message is seen in the server EQQMLOG:

```
EQQZ045E subtask server ended unexpectedly
```

In the output of the server task there was a SYSOUT with a TCPIP trace- entries with REQUEST HCREATE. These entries are created if the TCPIP DATA file has

the SOCKDEBUG parameter coded. Removing the SOCKDEBUG parameter from the system TCPIP DATA file prevents the abends from occurring.

18.9.8 Underscore "_" in DOMREC may cause IKJ56702I error

This is an example of the message:

```
IKJ56702I INVALID DOMAIN, TWS_DOMAIN_A
```

When defining a Tivoli Workload Scheduler for z/OS end-to-end scheduling DOMREC init statement, message IKJ56702I is received on startup of the E2E SERVER STARTED TASK, or on submission of a CP EXTEND, CP REPLAN or SYMPHONY RENEW batch planning job, if the DOMREC DOMAIN() keyword is coded using an UNDERSCORE in the DOMAIN NAME. The *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265 guide states that a DOMAIN NAME in the end-to-end scheduling topology may be up to 16 alphameric characters in length, must start with a letter, and may contain dashes (-) and underscores (_). However, the z/OS IKJPARS macro used to parse z initialization statements considers alphameric to include UPPER CASE CHARACTERS A-Z, numbers 0-9, and the national characters @, $, and #. It is not possible to tell this macro to allow the UNDERSCORE character without specifying ANY in the macro invocation, indicating it should pass ALL PRINTABLE CHARACTERS.

APAR PK17385 with fixing PTFs UK10792 for 8.1 and UK10793 for 8.2 changes the code to allow all (JCL-VALID) printable characters in the DOMREC DOMAIN() keyword, and the Tivoli Workload Scheduler manuals are updated to state that only characters A-Z, 0-9, -, and _ will be accepted by end-to-end scheduling processing, or by FTAs.

Users should apply the appropriate fixing PTF for APAR PK17835, and be careful when defining their end-to-end scheduling topology to make certain their chosen DOMAIN NAMES are limited to 16 characters in length and contain only the valid characters A-Z, 0-9, -, and _.

18.9.9 Message EQQPT60E and AWSEDW026E

The combination of this message in the E2E Server EQQMLOG:

```
EQQPT60E An END-TO-END process could end abnormally
```

along with the following messages in the USS Netman log can have different causes.

Example 18-74 Messages in the USS Netman log

```
AWSEDW026E Error firing child process: Service 2002 for TWS_z/OS/8.2 on
OPCMASTER(UNIX)
AWSDCJ010E Error opening new stdlist, Error: EDC5123I Is a directory."
```

One of them can be the incorrect specification of the BINDIR keyword in the TOPOLOGY statement. If, for example, BINDIR('usr/lpp/TWS/V8R2M0') is set (note that the first slash is omitted in the path) and the TWS server is started under a user having the root directory as HOME variable, some end-to-end scheduling processes can be started but the product does not behave as expected. Check that the path specified in the BINDIR keyword is an absolute path (i.e. it must begin with a slash).

18.9.10 Controller displays residual FTA status (E2E disabled)

If end-to-end scheduling is active on a Tivoli Workload Scheduler for z/OS Controller, then is disabled by removing the TPLGYSRV() keyword from the Controller OPCOPTS initialization statement, the Controller may continue to show FTA workstations as LINKED and/or ACTIVE.

The Controller continues to display the most recently received status information from all workstations. If end-to-end scheduling is disabled, the Controller will not receive any further information from the fault-tolerant workstations (FTAs), thus it continues to show the last status it received from those workstations before end-to-end scheduling processing was disabled. This is “as-designed” operation. It would not be correct to force all FTA workstations to INACTIVE status in the Controller displays, because the fault-tolerant nature of the FTAs means that after they have received a SYMPHONY file, they will continue to execute the work scheduled in that file even though they may no longer be able to communicate with the Controller (OPCMASTER).

18.10 Other useful end-to-end scheduling information

In the following sections, we provide some useful information that does not fit into any specific troubleshooting category.

18.10.1 End-to-end scheduling serviceability enhancements

PK01415, along with related APARs PK11095,PK11182, and PK11351 greatly improves the serviceability of the end-to-end scheduling environment. The following areas are addressed by PK01415:

1. Need CEEDUMP (LE) information in end-to-end scheduling SYSMDUMP

2. Lack of Problem Determination information during the daily planning phase
3. Lack of Problem Determination information when a file corruption occurs
4. USS file corruptions or file contention when multiple AS (address spaces) are generated for the server
5. Wrong definition of server and daily planning batch job users and groups
6. Bad process restart policy when an abend occurs

For more details, after installing the PTF for PK01415, download the EQQPDFST member from SEQQMISC to a workstation with an extension of .pdf, and open the file with Adobe Acrobat Reader.

18.10.2 Restarting an FTW from the distributed side

To restart an FTW (E2E FTA) from the distributed side without issuing commands from OPCMASTER, use one of the following examples to issue a link to the manager. These examples are for the domain manager FTA.

- Example 1:

```
StartUp
conman start
conman link masterdm!opcmaster
```

- Example 2:

```
StartUp
conman start
conman link @!@;noask
```

- Example 3:

```
StartUp
conman start
conman link @!opcmaster
```

18.10.3 Adding or removing an FTW

Use the following procedure to add a new FTW:

1. Define the new workstation through the Tivoli Workload Scheduler for z/OS dialog (panel 1.1.2) with this option:

```
FT WORK STATION=Y)
```

2. Add a CPUREC definition to the parmlib member specified in the TPLGYMEM parameter of the TOPOLOGY statement.
3. Run a CP REPLAN or EXTEND job to put the new workstation into effect.

4. It is *not* necessary to cycle the E2E Server task or Controller.

Use the following procedure to remove an FTW:

1. Delete the workstation from Tivoli Workload Scheduler for z/OS dialog (panel 1.1.2).
2. Remove the CPUREC definition from the parmlib member specified in the TPLGYMEM parameter of the TOPOLOGY statement.
3. Run a CP REPLAN or EXTEND job to put the workstation change into effect.
4. It is *not* necessary to cycle the E2E Server task or Controller.

18.10.4 Changing the OPCMASTER that an FTW should use

Sometimes, it is desirable to move an FTA that has been running with one OPCMASTER (such as a test controller) to a different OPCMASTER (for example, a production controller). Due to fault tolerance and the way that the Symphony file is built, the changes required to do this are not obvious. By default, the FTA will continue to receive the Sinfonia file from the original OPCMASTER. Use the following procedure for the domain manager and any FTAs that are to be moved from (for example) the old *test topology* to the new *production topology*:

1. Log on to the machine where the domain manager or FTA is installed, and stop Tivoli Workload Scheduler. (Bring Tivoli Workload Scheduler down with **`unlink`**, stop, and then stop netman with **`conman shut wait`**.)
2. Delete the existing Symphony and Sinfonia files and the *.msg and pobox/*.msg files. (That is, remove any memory of the old domain manager so that it will accept the new Sinfonia from the new OPCMASTER).
3. Modify the globalopts file located in the mozart directory.
4. For the domain manager only, change the NM Port value for the different OPCMASTER.
5. Restart Tivoli Workload Scheduler on the domain manager and FTAs to make the change take effect.
6. Cycle OPCMASTER (Tivoli Workload Scheduler z/OS Controller and its E2E Server task).

If the two OPCMASTERs are using the same nm port, the last one to pass a Symphony file will win (that is, its Sinfonia will be the one used). If any FTA will remain in the old topology but will *not* be in the new topology, you must make certain it is STOPPED. Otherwise, the PDM and its old network of FTAs will continue to try to process the Symphony file they received from your test Controller, due to fault tolerance.

18.10.5 Reallocating the EQQTWSIN or EQQTWSOU file

Sometimes, it is necessary to reallocate the EQQTWSIN and EQQTWSOU files (for example, if they have to be increased in size). When reallocating the EQQTWSIN/EQQTWSOU data sets, it is important that no unprocessed events are present in the data sets. Loss of unprocessed events might cause subsequent problems. Follow this procedure to avoid the loss of events:

1. Wait for all operations running on FT workstations to be completed.
2. Be sure that no DP batch is running (CP EXTEND, CP REPLAN, or SYMPHONY RENEW).
3. Stop the Controller and E2E Server.
4. Allocate a larger data set for the specified ddname (using the EQQPCS06 job).
5. Restart the Controller and E2E Server.
6. When the Controller is restarted, it will FORMAT the new data set. Do *not* copy the contents of the old data set into the new one. Just allow the Controller to initialize it.

18.10.6 E2E Server SYSMDUMP with Language Environment (LE)

Note: If PK01415 is applied, this section can be ignored (no longer needed).

Sometimes, it is necessary to capture a dump of the E2E Server that includes LE information. The easiest way to do this is to include the LE STDENV variables in a parmlib member allocated to the E2E Server task as STDENV. A SYSMDUMP data set must also be allocated to the server task as shown in Example 18-75. The SYSMDUMP file should be allocated at 200 cylinders or more.

Example 18-75 Include the LE STDENV variables in a parmlib member

```
//SYSMDUMP  DD  DISP=MOD,DSN=tws.e2e.sysmdump
//STDENV    DD DISP=SHR,DSN=TWSZ..PARMLIB(ENV)
```

The contents of the parmlib member ENV will be a single line that refers to the SYSMDUMP data set name:

```
_BPXK_MDUMP=tws.e2e.sysmdump
```

After the server task is started with these changes in place, it is necessary to determine the process ID (PID) of the translator process. This can be done by

looking at the server EQQMLOG for message EQQPT01I, where bindir is the bin directory (eqqBINDIR):

```
EQQPT01I Program "/bindir/bin/translator" has been started, pid is
nnnnnnnn
```

The EQQPT01I message is also issued for the netman process. Be sure to use the translator PID. The other way to get the PID for the translator is to issue the following command:

```
D OMVS,A=ALL
```

The preceding command displays information for all OMVS processes. By checking the hexadecimal ASID value (ASIDX) for the E2E Server in SDSF, the following command can be issued to display information only for the processes owned by the E2E Server task:

```
D OMVS,A=asid
```

(In this command, `asid` is the hexadecimal ASID of the server address space.)

Example 18-76 shows an example of a display of the OMVS processes for a E2E Server task.

Example 18-76 Example of a display of the OMVS processes for an E2E Server task

```
BPXO040I 20.05.52 DISPLAY OMVS 044
 OMVS     000D ACTIVE         OMVS=(01,0A,00)
 USER     JOBNAME  ASID       PID      PPID STATE   START     CT_SECS
 O82STSO  O82S     003B   50331709   67109052 1F---- 20.03.13      .58
   LATCHWAITPID=        0 CMD=/u/u/usr/lpp/TWS/V8R2M0/bin/netman
 O82STSO  O82S     003B   67108957         1 MW---- 20.03.12      .58
   LATCHWAITPID=        0 CMD=EQQPHTOP
 O82STSO  O82S     003B       126   67109046 1F---- 20.03.25      .58
   LATCHWAITPID=        0 CMD=/u/u/usr/lpp/TWS/V8R2M0/bin/batchman
 O82STSO  O82S     003B   16777357   67109052 HS---- 20.03.13      .58
   LATCHWAITPID=        0 CMD=/u/u/usr/lpp/TWS/V8R2M0/bin/translator
 O82STSO  O82S     003B   67109046   50331709 1F---- 20.03.24      .58
   LATCHWAITPID=        0 CMD=/u/u/usr/lpp/TWS/V8R2M0/bin/mailman
 O82STSO  O82S     003B   67109052   67108957 1S---- 20.03.12      .58
   LATCHWAITPID=        0 CMD=/u/u/usr/lpp/TWS/V8R2M0/bin/starter
```

In this example, 16777357 was the PID of the translator process. After the PID of the translator process has been determined by either method, issue the following command to capture the dump with LE information:

```
F BPXOINIT,DUMP=translator_pid
```

For the values displayed above, the command would be:

```
F BPXOINIT,DUMP=16777357
```

The dump will be shown as SEC6 abend in the server address space JESMSGLG, as shown in Example 18-77.

Example 18-77 Excerpt from JESMSGLG

```
IEA995I SYMPTOM DUMP OUTPUT  083
SYSTEM COMPLETION CODE=EC6  REASON CODE=0D2FFD27
```

18.10.7 Analyzing file contention within the E2E Server

In order to resolve file contention in the address space of the E2E Server, determine which process has the file in question open. Because the E2E Server is a application running in z/OS UNIX (formerly UNIX System Services), a useful command is **fuser**.

Resource contention is common, when multiple programs, processes, and so forth try to use the same resource. If the lock is released in an acceptable time, there is no need to take care of it. If the lock lasts too long or is not released, it can cause severe problems and must be analyzed.

Command **fuser** returns the PID (program ID) of every process that has the file open. The -u option returns the UID of the process in addition. To send a SIGKILL command to every process in order to release the resources, this command can be used:

```
fuser -kuc
```

For a detailed description of **fuser** use the man pages command from the shell:

```
man fuser
```

18.10.8 Determining the fix pack level of an FTA

The fix pack level used by an FTA does not show up in the FTA logs. To determine the level of fix pack installed on a Tivoli Workload Scheduler fault-tolerant agent, look in the directory <twshome>\version for a file named patch.info.

The content of this file is similar to the following line, which indicates that Fix Pack 02 has been installed on Release 8.2:

```
8.2-TWS-FP02
```

When the machine is unpatched (GA version), the version directory will be empty.

18.11 Where to find messages in UNIX System Services

UNIX System Services has a daemon, called the *syslogd* daemon, that is responsible for the logging of UNIX System Services (USS) messages. The daemon can be started in several different ways. One is during IPL through the /etc/rc script, as shown in Example 18-78.

Example 18-78 Starting syslogd in the /etc/rc script

```
_BPX_JOBNAME='SYSLOGD' /usr/sbin/syslogd -f /etc/syslog.conf
```

Another way to start it uses a cataloged procedure as shown in Example 18-79.

Example 18-79 SYSLOGD procedure

```
//SYSLOGD  PROC
//SYSLOGD  EXEC PGM=SYSLOGD,REGION=30M,TIME=NOLIMIT
//       PARM='POSIX(ON) ALL31(ON)/ -f /etc/syslogd.conf'
//SYSPRINT    DD SYSOUT=*
//SYSIN       DD DUMMY
//SYSOUT      DD SYSOUT=*
//SYSERR      DD SYSOUT=*
//CEEDUMP     DD SYSOUT=*
```

The syslogd configuration file etc/syslogd.conf contains the definitions where the types of messages are routed. Example 18-80 shows an example.

Example 18-80 The syslogd configuration file

```
# all alert messages (and above priority messages) go to the MVS consol
*.alert                 /dev/console
#
# all authorization messages go to auth.log
auth.*                  /var/log/%Y/%m/%d/auth.log
#
# all error messages (and above priority messages) go to error.log
*.err                   /var/log/%Y/%m/%d/error.log
#
# all debug messages (and above priority messages) from telnet
# from telnet go to local1.debug
local1.debug            /var/log/%Y/%m/%d/local1.debug
```

As you can see in Example 18-80, alert messages are issued at the console, and Telnet debug messages, for example, are routed to the /var/log/ directory. Remember that messages can be located in different places.

Failures and messages in the z/OS UNIX environment

Hard failures end with an abend or message from the failing function including return and reason codes. For a description of abend codes, refer to *z/OS MVS System Codes*, SA22-7626.

z/OS USS messages and codes

Table 18-1 shows a list of message identifiers in z/OS UNIX.

Table 18-1 Message identifiers and related components

Identifier	Component or message type
BPX	z/OS USS MVS system messages
FSUM	USS shell and utilities messages
FDBX	USS debugger (DBX) messages
FOM	USS application services messages
EDC	LE C/C++ runtime library (RTL) messages
CEE	LE base messages
CBC	C/C++ compiler messages
EZx	TCP/IP messages
ICH/IRR	RACF messages
IMW	WebSphere® messages

To list messages online, you can use the LookAt message tool, available at:

`http://www.ibm.com/servers/s390/os390/bkserv/lookat/lookat.html`

Messages from failing z/OS UNIX functions

Description of the UNIX return and reason codes are available in *z/OS UNIX System Services Messages and Codes*, SA22-7807.

z/OS UNIX return codes generally correspond to standard POSIX errors such as EAGAIN (resource is temporarily unavailable), EACCESS (is denied), and EBUSY (resource is busy).

z/OS UNIX reason codes, also referenced as *errnojr*, are made up of 4 bytes in the following format:

```
cccc rrrr
```

The cccc is a reason code qualifier. This is used to identify the issuing module and represents a module ID. The second 2 bytes are the reason codes that are described in the messages books.

If this value is between 0000 and 20FF, and the return code is not A3 or A4, this is a USS reason code. In this situation, you can use the BPXMTEXT procedure to get more information. See also 18.1.1, “EQQISMKD” on page 610 for an example of the BPXMTEXT command.

18.12 Where to find messages in an end-to-end environment

There are several places where messages pertaining to an end-to-end scheduling issue might be stored. These include:

- The E2E Server EQQMLOG.
- The Tivoli Workload Scheduler for z/OS Controller EQQMLOG.
- The EQQMLOG of the CP batch job (EXTEND, REPLAN, or SYMPHONY RENEW).
- The USS WRKDIR in directory /<eqqWRKDIR>/stdlist/logs, which has the following files for each day (where yyyymmdd is the date format):
  - yyyymmdd_NETMAN.log
  - yyyymmdd_TWSMERGE.log
  - yyyymmdd_E2EMERGE.log
- For each FTA or DM, there are logs in the <HOME DIRECTORY>/stdlist/logs directory:
  - yyyymmdd_NETMAN.log
  - yyyymmdd_TWSMERGE.log
- The MVS SYSLOG might contain useful messages if you issued any commands to OMVS, such as:
  - D OMVS,A=ALL
  - D OMVS,O
  - D OMVS,L

It is important for the EQQMLOGs to ensure that these files are not purged or overwritten. For example, if you write the EQQMLOGs to SYSOUT, either keep them on the spool or use your SYSOUT archival product to store them. If you write the EQQMLOGs to disk, insert a step in the startup JCL for the task to save the previous EQQMLOG to a generation data group (GDG) or other data set before the step that would overwrite the EQQMLOG file.

A

Version 8.2 PTFs and a Version 8.3 preview

In this appendix we first present the Tivoli Workload Scheduler for z/OS V8.2 PTFs, then give a preview of what is coming in Tivoli Workload Scheduler for z/OS V8.3, which is currently scheduled to be available in 4Q 2006.

Important: Because Tivoli Workload Scheduler for z/OS V8.3 is not available yet, all planned enhancements that are mentioned here are subject to change.

Tivoli Workload Scheduler for z/OS V8.2 PTFs

Tivoli Workload Scheduler V8.2 has had multiple fixes over the past year. The list of the problems and the resolutions, along with the PTF numbers, includes:

- In a end-to-end environment, a job is defined with a scriptlib member containing errors, the daily planning batch job sets the status of that job in fail status in the Symphony file. Messages explaining the error can be found in the Daily Planning log. Also in the current plan, if a job is dynamically added and the associated script member contains errors, the job is not added. Messages explaining the problem can then be found in the log of the Controller. This causes problems in creating or executing the daily plan and can cause business problems.
  - PQ99317 resolves this problem by first having a Trial plan created before running the current plan, so that it reveals errors in the SCRPTLIB members. Error messages are issued in the EQQDTTOP mlog, in order to take the correct actions to avoid problems when the daily plan is created.
  - The EQQSLCHK sample JCL provides a syntactic check of all script library members. This sample will run stand-alone, without the need to interact with the CP database. It can also be run against all script members in the SCRPTLIB or can be used to perform a check against only a subset of these script members.
- Currently, when a server started task abends in the C/C++ code, a CEEDUMP (Language Environment® Dump) of the original abend and a SYSMDUMP with completion code U4039 are taken. The CEEDUMP contains partial data related to the address space and is not enough for a complete error analysis. The SYSMDUMP sometimes is not related to the original abend. So there must be a dump of the original abend as a SYSMDUMP.
  - In PTF PK01415 the SYSMDUMP of the original abend is now collected containing also the LEDATA and CEEDUMP information.
- Lack of problem determination information and messages during the daily planning phase.
  - In PTF PK01415 the new trace and log messages have been added to analyze errors occurred during the daily planning phase.
- Lack of problem determination information when a file corruption occurs.
  - In PTF PK01415 to intercept errors corrupting or locking the event files, we have traced open and write calls on them. In this way it will be easier to understand the origin of the corruption. This trace will write the messages in memory in order to avoid filling log files and possible performance reductions. If an error occurs while accessing an event file a dump is taken for problem determination purposes; the memory traceinfo will be available in the dump.

- USS file corruptions or contentions when multiple AS (address space) are generated for the server. Customer experienced event file corruption and contention when the Tivoli Workload Scheduler server generated processes with parent process id (ppid) equal to 1 (for instance, Batchman process). The reason for this problem was the cancellation of the server started task when multiple address spaces had been generated; only one address space must be generated for all the server tasks/processes/threads.
  - PTF PK01415 offers the following solution. Avoiding USS file corruptions and contentions generated by multiple AS for the server. To avoid the generation of multiple address spaces, the following actions are performed: 1) Rework of the environmental variables; in particular the _BPX_SHAREAS variable is now always set to YES for all the processes and threads. 2) Batchman, mailman, and writers processes have the same PGID of the netman process.
- Wrong definitions of server and Daily planning batch job users and groups. Wrong definitions of RACF users and groups, even concerning user IDs not related to the end-to-end server, can cause errors in the Server or Daily Plan batch processing that sometimes does not have a direct relationship with the original error on the users/groups.
  - PTF PK01415 resolution checks on users and groups definitions in the RACF database have been introduced:

    The server, at startup and every five minutes, checks:

    - User assigned to the server
    - UID of this user
    - All users with UID equal to the user assigned to the server
    - Group for every user found
    - GID for every group

    Every user or group that does not have an OMVS segment assigned is reported in EQQMLOG with an error message. Error messages are issued only once, even if the related problem has not been solved. When the problem is solved, a new message is issued to the EQQMLOG. If there is an RACF access error, the problem is reported with a warning message in EQQMLOG.
  - PTF PK01415. Daily Plan batch, at startup, makes the same checks as the server, but only once. If the user starting Daily Plan batch has no UID or the default group has no GID, Daily Plan batch is stopped and it ends with RC=12. If there are users with the same UID of user starting Daily Plan batch, and one of them has a default group with no GID defined, Daily Plan batch is stopped; Daily Plan batch such as current plan extension, replan, and trial return RC=12, while Symphony renew returns RC=8.

- Bad processes restart policy when an abend occurs. In case of fatal errors (abends), the Starter process restarts its children indefinitely.
  - PTF PK01415. New processes restart policy when an abend occurs.

    TRANSLATOR and NETMAN processes policy. In case of error, at start up or at run time, on the translator or netman processes, the starter tried to restart them indefinitely. Now the restart process has been changed in the following way:

    i. If translator goes down then starter tries to restart it after no more than 5 minutes.
    ii. If netman goes down, mailman, batchman and writers also go down. Because translator is strictly related to batchman and mailman, translator also goes down. Also in this case starter tries to restart netman and translator after no more than five minutes.
    iii. Now Starter tries to restart translator and netman just once; but if an abend occurs after more than two hours since the last process restart, a new restart is attempted. If the problem persists, then message EQQPT63E is logged and the starter closes.
  - PTF PK01415. The MAILMAN and BATCHMAN processes policy. In case of an error (abend) in the mailman or batchman processes, the following message will be printed in the MLOG: `EQQPT33E Mailman or Batchman ended abnormally.` Translator and his children begin to shut down. After that the translator goes down and then its restart policy applies.
  - In PTF PK01415 the following minor problems have been fixed:
    - If SERVOPT PROTOCOL (End-to-End) is specified and the starter process abends, now the whole server's started task closes.
    - The Trial Current Plan does not end with RC=0 any more, even if there is an error in the domains and workstations definitions.
    - If EQQ3041W is issued during the symphony renew, the EQQ3088E is no longer issued twice and the batch does not end with return code zero.
    - EQQPT06I text has been corrected.
- In an end-to-end configuration, the output devices for the Server messages are: the server MLOG in this file are issued the EQQ messages coming from the Tivoli Workload Scheduler for z/OS and most EQQPT messages coming from the end-to-end environment. Also the stdlist directory in the wrkdir (E2EMERGE.log, TWSMERGE.log, and NETMAN.log) in these files are issued all EQQPT messages coming from the end-to-end environment and all AWS messages coming from the Tivoli Workload Scheduler processes.
  - APAR PK11341 provides the capability to send a customizable set of Tivoli Workload Scheduler for z/OS Server messages (EQQPT) and Tivoli

Workload Scheduler messages (AWS) to the server MLOG and to the System Output Console (SYSLOG).

- End-to-end reduce network shutdown.
  - APAR PK11811 improves the scheduling system by reducing the “inactivity window” in the end-to-end environment.
- End-to-end server messages were not written to SYSLOG.
  - APAR PK11314 End-to-end Server messages are not written to SYSLOG to allow automatic detection of end-to-end server messages via system automation tools.
- Tivoli Workload Scheduler for z/OS Workload Manager integration requirement.
  - APARs PK08998, PK08999, and PK09001 provide Tivoli Workload Scheduler for z/OS Workload Manager integration based on scheduling environments. This enables optimal use of available resources and to ensure that you are getting the most from your existing resources and that the Workload between resources is automatically balanced for optimal throughput and performance.

Preview of Tivoli Workload Scheduler for z/OS V8.3

The following enhancements are planned for the next release of Tivoli Workload Scheduler for z/OS Version 8.3, which is scheduled to be available in 4Q 2006.

- Enhance variables support in event-triggered jobs. This way there will be no need to manually tailor event-triggered jobs.
- Increase capability to schedule unplanned workload, via Event Trigger Tracking (ETT). This will provide additional flexibility during event-driven planning.
- Improve special resources flexibility to simulate file dependencies. That way there will be no need to manually change Special Resource availability status in customer business process scenarios. This will reduce the Total Cost of Ownership to implement common scenarios.
- Critical paths will be determined by Tivoli Workload Scheduler for z/OS to user-defined critical jobs. Operations on critical path will be automatically promoted (via operation internal priority and WLM high-performance service classes), to reduce operator manual intervention.
- Introduce a delay between jobs to reduce Total Cost of Ownership in implementing a solution to delay job submission respect to predecessor completion.

- Implement every option in application run cycle. This will help to reduce the cost to create and maintain workflows, improve planning flexibility, and reduce the cost to migrate from other scheduling products.
- Data storing capacity improvement will improve scalability for Restart and Cleanup function in multiple environments.
- Enhanced end-to-end standard agent connected to master domain manager. This will create centralized control of jobs running on standard agents.
- Integration with Tivoli Enterprise Portal. This will permit you to monitor Tivoli Workload Scheduler for z/OS data through a common and consistent interface.
- A new enhancement, the ability to simplify the pinpoint of loop conditions and improve performance of loop detection. This will improve performance in Tivoli Workload Scheduler for z/OS loop dependencies analysis and greatly reduce Total Cost of Ownership for identifying the loop's real cause.

B

EQQAUDNS member example

This appendix provides an example of the EQQAUDNS member that resides in the HLQ.SKELETON DATASET. Refer to discussion about this topic in 1.5.3, “Option 2” on page 19.

An example of EQQAUDNS member that resides in the HLQ.SKELETON DATASET

Example: B-1 EQQAUDNS member

```
PROC 0
  CONTROL NOLIST NOCONLIST NOMSG
  /*****************************************************************  */
  /*   THIS HAS BEEN MODIFIED FOR EXPLANATION PURPOSES OF THIS REDBOOK */
  /*   A /* <<<<<<< */ COMMENT HAS BEEN ADDED TO INDICATE             */
  /*   THE ALLOCATION NAMES THAT SHOULD BE REVIEWED FOR YOUR SYSTEM    */
  /*****************************************************************/
  /*   Licensed Materials - Property of IBM                        */
  /*   5697-WSZ                                                    */
  /*   (C) Copyright IBM Corp. 1990, 2001 All Rights Reserved.     */
  /*   US Government Users Restricted Rights - Use, duplication    */
  /*   or disclosure restricted by GSA ADP Schedule Contract       */
  /*   with IBM Corp.                                              */
  /*****************************************************************/
  /* $BBL=PQ61882 020604 SV: RACF violation if TSO user has NOPROFILE*/
  /* $BKR=PQ71640 030304 SV: Sysroute of previous to V8            */
  /* $BLE=PQ72132 030402 SV: Ignored options entered at batch submis.*/
  /* $BQV=PQ81135 031120 TM: Chg Sysin file in EQQAUDNS            */
  /*                                        02 lines changed $BQVC*/
  /* $BTO=PQ84010 040311 TM: EQQJOBS fails with line length error in */
  /*                         skeleton EQQAUDNZ (04 lines erased).    */
  /*****************************************************************/
  /*                                                               */
  /*  THIS CLIST WILL CALL AN ISPF PANEL WHICH WILL ASK FOR PARMS TO */
  /*  PASS TO LOG RECORD EXTRACT/FORMAT SAMPLE PROGRAM EQQAUDIT.     */
  /*  NEEDED OUTPUT FILES WILL BE ALLOCATED, THE PGM WILL BE CALLED, */
  /*  AND THE RESULTING REPORT WILL BE DISPLAYED.                    */
  /*                                                               */
  /*****************************************************************/
  /*ATTENTION                                                      */
  /* The output of the processing of this skeleton is EQQAUDNS:     */
  /* it MUST be copied in the procedure library WITHOUT ANY         */
  /* FURTHER CUSTOMIZATION if proper input supplied at              */
  /* installation time.                                            */
  /* This CLIST is used in the interactive invocation of           */
  /* EQQAUDIT.                                                     */
  /*ATTENTION                                                      */
  /* DISPLAY PANEL TO COLLECT RUN PARAMETERS FOR EQQAUDIT        */
  /*--------------------------------------------------------------*/
```

```
ISPEXEC DISPLAY PANEL(EQQAUDIP)
SET DISPCC = &LASTCC
IF &DISPCC = 0 THEN +
  DO
    /*-------------------------------------------------------------*/
    /* FIRST GET ALL THE PARMS ENTERED ON THE PREVIOUS PANEL    */
    /*-------------------------------------------------------------*/
    ISPEXEC VGET (AUI AUSTRING AUSD AUST AUED AUET) PROFILE
    ISPEXEC VGET (AUDOUTDS) PROFILE             /*         @57A*/
    /*-------------------------------------------------------------*/
    /* NOW GET ALL THE VARIABLES THAT MIGHT REMAIN IN THE   @57A*/
    /* EQQAUDNS UNRESOLVED                                 @57A*/
    /*-------------------------------------------------------------*/
    ISPEXEC VGET (XOPCNM) PROFILE                      /* @57A*/
    /*-------------------------------------------------------------*/
    /*-------------------------------------------------------------*/
    /* THEN ALLOCATE ALL NEEDED FILES                           */
    /*-------------------------------------------------------------*/
    FREE  F(SYSPRINT,SYSIN)                            /* @01C*/
    FREE  F(EQQPARM,EQQMLIB,EQQMLOG,LIVEMLOG,AUDITPRT)
    ALLOC F(SYSPRINT) DA(*)
                                                       /* @01D*/
    CONTROL NOFLUSH
    ALLOC F(SYSIN) +
          DA('&SYSUID..EQQAUDIT.SYSIN') +
          SHR REU
    SET RCODE = &LASTCC
    CONTROL FLUSH NOMSG
    IF &RCODE ¬= 0 THEN +
      DO
        ALLOC FI(SYSIN) +
              DA('&SYSUID..EQQAUDIT.SYSIN') +
              NEW TRACKS SPACE(1,1) CATALOG REUSE  +
              RECFM(F B) LRECL(80) +
              BLKSIZE(800) UNIT(3390)
        WRITE SYSIN-FILE DID NOT EXIST, HAS BEEN ALLOCATED
      END
    OPENFILE SYSIN OUTPUT
    SET L = &LENGTH(&STR(&AUSTRING))      /* FIGURE OUT LENGTH  */
    IF &L ¬= 16 THEN +
      DO
        SET F = (16 - &L)                   /* LENGTH OF FILLERS  */
        SET F = &SUBSTR(1:&F,&STR(                ))
        SET AUSTRING = &STR(&AUSTRING&F)  /* ADD TRAILING BLANKS*/
      END
```

```
IF &AUSD = &STR() THEN +
  SET AUSD = &STR(      )
IF &AUST = &STR() THEN +
  SET AUST = &STR(    )
IF &AUED = &STR() THEN +
  SET AUED = &STR(      )
IF &AUET = &STR() THEN +
  SET AUET = &STR(    )
SET SYSIN  = &AUI&AUSTRING&AUSD&AUST&AUED&AUET
PUTFILE SYSIN
CLOSFILE SYSIN
ALLOC F(EQQPARM) DA('HLQ.PARM(OPCA)') +                  /* <<<<<<< */
  SHR REU
ALLOC F(EQQMLIB) DA('HLQ.SEQQMSGO') +                    /* <<<<<<< */
  SHR REU
/*-------------------------------------------------------------*/
/* FILE BELOW IS THE MLOG WRITTEN BY THE EQQAUDIT ITSELF     */
/*-------------------------------------------------------------*/
ALLOC F(EQQMLOG) DUMMY
/*-------------------------------------------------------------*/
/* FILE BELOW IS THE MLOG WRITTEN BY THE CONTROLLER      @57M*/
/*-------------------------------------------------------------*/
ALLOC F(LIVEMLOG) DUMMY
CONTROL NOFLUSH
/*-------------------------------------------------------------*/
/* FILE BELOW IS WHERE THE REPORT IS WRITTEN             @57C*/
/*-------------------------------------------------------------*/
IF &AUDOUTDS = &STR() THEN +
  ALLOC F(AUDITPRT) +
        DA('HLQ.EQQAUDIT.REPORT') +                      /* <<<<<<< */
        SHR REU
ELSE +
  ALLOC F(AUDITPRT) DA('&AUDOUTDS.') +
    SHR REU
SET RCODE = &LASTCC
CONTROL FLUSH NOMSG
IF &RCODE ¬= 0 THEN +
  DO
    ALLOC F(AUDITPRT) +
          DA('&AUDOUTDS.') +
          NEW TRACKS SPACE(20,50) CATALOG REUSE +
          RECFM(F B A) LRECL(133) +
          BLKSIZE(13300) UNIT(3390)
    WRITE LIST-FILE DID NOT EXIST, HAS BEEN ALLOCATED
  END
```

```
IF &AUI = JTX THEN +
  DO
    /*-------------------------------------------------------*/
    /* FILES BELOW ARE THOSE SPECIFIED IN STC-JCL FOR THE     */
    /* CONTROLLER SUBSYSTEM AND USED IF INPUT OPTION IS 'JTX'.*/
    /* THE NEED TO INSERT MORE EQQJTxx DATASETS, IF       @57C*/
    /* THE NUMBER SPECIFIED IN JTLOGS(x) KEYWORD IN JTOPTS    */
    /* IS HIGHER THAN 2, HAS BEEN ELIMINATED.             @57C*/
    /*-------------------------------------------------------*/
    FREE  F(EQQJTARC,EQQJT01,EQQJT02,EQQCKPT)
    FREE  F(EQQJT03,EQQJT04,EQQJT05)              /*      @57A*/
    ALLOC F(EQQJTARC) DA('HLQ.JTARC') +                 /* <<<<<<< */
      SHR REU
    ALLOC F(EQQJT01)  DA('HLQ.JT1')   +                 /* <<<<<<< */
      SHR REU
    ALLOC F(EQQJT02)  DA('HLQ.JT2')   +                 /* <<<<<<< */
      SHR REU
    ALLOC F(EQQJT03)  DA('HLQ.JT3')   +                 /* <<<<<<< */
      SHR REU                                     /*      @57A*/
    ALLOC F(EQQJT04)  DA('HLQ.JT4')   +                  /* <<<<<<< */
      SHR REU                                     /*      @57A*/
    ALLOC F(EQQJT05)  DA('HLQ.JT5')   +                  /* <<<<<<< */
      SHR REU                                     /*      @57A*/
    ALLOC F(EQQCKPT)  DA('HLQ.CKPT')    +                /* <<<<<<< */
      SHR REU
  END
ELSE +
  DO
    FREE  F(EQQTROUT)
    /*-----------------------------------------------*/
    /* FILE BELOW IS CREATED IN DAILY PLANNING BATCH */
    /* AND USED IF INPUT OPTION  'TRL'          @57C*/
    /*-----------------------------------------------*/
    ALLOC F(EQQTROUT) DA('HLQ.TRACKLOG') +               /* <<<<<<< */
       SHR REU
  END
/*-------------------------------------------------------------*/
/* NOW INVOKE THE PROGRAM ITSELF                          @57C*/
/*-------------------------------------------------------------*/
WRITE **** NOW INVOKING EQQAUDIT. THIS MAY TAKE A WHILE ****
CALL '(EQQBATCH)' 'EQQAUDIT'
WRITE RETURN CODE FROM PGM WAS: &LASTCC
/*-------------------------------------------------------------*/
/* A REPORT FILE HAS BEEN PRODUCED, DISPLAY IT            @57C*/
/*-------------------------------------------------------------*/
```

```
      IF &AUDOUTDS = &STR() THEN +
        ISPEXEC BROWSE DATASET('HLQ.EQQAUDIT.REPORT')           /* <<<<<<< */
      ELSE +
        ISPEXEC BROWSE DATASET('&AUDOUTDS.')
      /*-------------------------------------------------------------*/
      /* FINALLY FREE THE FILES                                      */
      /*-------------------------------------------------------------*/
      FREE F(SYSPRINT,SYSIN)                                 /* @01C*/
      FREE F(EQQPARM,EQQMLIB,EQQMLOG,LIVEMLOG,AUDITPRT)
      IF &AUI = 'JTX' THEN +
       DO                                                   /*   @57A*/
        FREE F(EQQJTARC,EQQJT01,EQQJT02,EQQCKPT)
        FREE F(EQQJT03,EQQJT04,EQQJT05)                     /*   @57A*/
       END                                                  /*   @57A*/
      ELSE +
        FREE F(EQQTROUT)
    END
EXIT
```

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG247156

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds with the redbook form number, SG247156.

Using the Web material

The additional Web material that accompanies this book includes the following file:

File name	*Description*
SG247156.zip	E2E Big Poster, data set sizing spreadsheet and EQQAUDNS member example

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 10 MB minimum
Operating System: Windows/Linux/UNIX

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 682. Some of the documents referenced here may be available only in softcopy.

- *Customizing IBM Tivoli Workload Scheduler for z/OS V8.2 to Improve Performance*, SG24-6352
- *Integrating IBM Tivoli Workload Scheduler with Tivoli Products*, SG24-6648

Other publications

These publications are also relevant as further information sources:

- *IBM Tivoli Workload Scheduler for z/OS Installation Guide Version 8.2,* SC32-1264
- *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning Version 8.2,* SC32-1265
- *IBM Tivoli Workload Scheduler for z/OS Diagnosis Guide and Reference Version 8.2*, SC32-1261
- *IBM Tivoli Workload Scheduler for z/OS Managing the Workload Version 8.2,* SC32-1263
- *z/OS JES3 Initialization and Tuning Reference*, SA22-7550
- *IBM Tivoli Workload Scheduler Reference Guide,* SC32-1274
- *IBM Tivoli Workload Scheduler Job Scheduling Console Release Notes,* Feature level 1.4, SC32-1258
- *IBM Tivoli Workload Schedulers Job Scheduling Console V1.4 User’s Guide* SC32-1257

Online resources

This Web site is also relevant as an information source:

- IBM Tivoli Workload Scheduler for z/OS manuals

  http://publib.boulder.ibm.com/tividd/td/WorkloadScheduler8.2.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

IBM Tivoli Workload Scheduler for z/OS Best Practices

End-to-end and mainframe scheduling

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM Tivoli Workload Scheduler for z/OS Best Practices

End-to-end and mainframe scheduling

Redbooks

A guide for system programmers and administrators

Covers installation and customization

Includes best practices from the field

This IBM Redbook serves as a reference for system programmers and administrators who will be installing IBM Tivoli Workload Scheduler for z/OS in mainframe and end-to-end scheduling environments.

Installing IBM Tivoli Workload Scheduler for z/OS requires an understanding of the started tasks, the communication protocols and how they apply to the installation, how the exits work, how to set up various IBM Tivoli Workload Scheduler for z/OS parameters and their functions, how to customize the audit function and the security, and many other similar topics.

In this book, we have attempted to cover all of these topics with practical examples to help IBM Tivoli Workload Scheduler for z/OS installation run more smoothly. We explain the concepts, then give practical examples and a working set of common parameters that we have tested in our environment.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7156-01 ISBN 0738497002