Tivoli software

IBM

Implementing IBM Tivoli Workload Scheduler V8.2 Extended Agent for IBM Tivoli Storage Manager

- Insider's guide to Tivoli Workload Scheduler extended agents
- Ready-to-use solution for TSM and TWS integration
- TSM Extended Agent code included

Archived

Vasfi Gucer
John Ellery
Carl Buehler

Redbooks

ibm.com/redbooks

Archived

International Technical Support Organization

Implementing IBM Tivoli Workload Scheduler V8.2 Extended Agent for IBM Tivoli Storage Manager

May 2005

Archived

SG24-6696-00

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

Archived

First Edition (May 2005)

This edition applies to IBM Tivoli Workload Scheduler Version 8.2, IBM Tivoli Storage Manager Version 5.3.

© Copyright International Business Machines Corporation 2005. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2005. All rights reserved.

Archived

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.

© Copyright IBM Corp. 2005. All rights reserved.

Trademarks

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

AIX®	MVS™	RS/6000®
@server®	Redbooks™	Tivoli®
IBM®	Redbooks (logo) ™	

The following terms are trademarks of other companies:

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

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

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

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

Archived

Preface

IBM® Tivoli® Workload Scheduler is a strategic, multiplatform distributed scheduling product that provides high-volume, complex scheduling capability. Although Tivoli Workload Scheduler provides native support for many platforms and applications, its robust scheduling capabilities can be extended to cover additional platforms and applications by writing an extended agent.

This IBM Redbook shows how to write a Tivoli Workload Scheduler Version 8.2 extended agent to schedule jobs on IBM Tivoli Storage Manager. With the extended agent, you can schedule on platforms and applications for which Tivoli Workload Scheduler has no native agent, as well as integrate IBM Tivoli Storage Manager with Tivoli Workload Scheduler. The Tivoli Workload Scheduler scheduling facility enables you to assign dependencies among tasks scheduled through Tivoli Storage Manager or to assign limits or priorities. By extending Tivoli Storage Manager to schedule these Tivoli Storage Manager tasks, you can take advantage of its advanced scheduling capabilities.

This book will be essential for those who write a Tivoli Workload Scheduler extended agent for any platform in general, or use the extended agent provided in this book (TSM Extended Agent) to schedule Tivoli Storage Manager tasks.

The team that wrote this redbook

This redbook 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.

John Ellery is a Senior IT Consultant with IBM Business Partner Automatic IT Corporation, http://www.AutomaticIT.com, in Austin, Texas. His areas of expertise include Tivoli Workload Scheduler for distributed and end-to-end environments, as well as the integration of Tivoli Workload Scheduler with other products and platforms. He has delivered Tivoli Workload Scheduler training, implementation, customization, and migration services at more than 50 companies during the past eight years. Before that, he worked for 13 years in the

© Copyright IBM Corp. 2005. All rights reserved.

defense industry as a software developer on military weapon systems design projects.

Carl Buehler is an IBM Certified IT Specialist working in IBM Software Services for Tivoli (ISST). He joined IBM in 1987 and has been a software developer, software support engineer, and technical marketing demo developer in addition to his current role in services. He has worked with Tivoli Workload Scheduler since the ESP for Version 8.2 in 2003 and has extensive experience deploying the product with customers.

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

Betsy Thaggard
International Technical Support Organization, Austin Center

We also thank the authors of the redbook *Implementing TWS Extended Agent for Tivoli Storage Manager*, SG24-6030 (based on IBM Tivoli Workload Scheduler V7.0). which the redbook updates for Tivoli Workload Scheduler V8.2 and the JSC user interface.

Henry Daboub, Warren Gill, and Tina Lamachia
IBM USA

Denise Kikumoto
IBM Brasil

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:

http://www.redbooks.ibm.com/residencies.html

Archived

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:

  `http://www.redbooks.ibm.com`

- Send your comments in an e-mail to:

  `redbook@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. JN9B Building 905
  11501 Burnet Road
  Austin, Texas 78758-3493

Archived

Archived

1

Introduction

This redbook is designed to illustrate the implementation of a Tivoli Workload Scheduler (TWS) extended agent. The implementation the redbook team designed facilitates a communication between Tivoli Workload Scheduler V8.2 and Tivoli Storage Manager (TSM) V5.3 to initiate regularly scheduled tasks on the TSM server. The TSM server-prompted scheduling facility enables us to initiate client backups on machines defined as TSM clients in server-prompted scheduling mode.

Archived

© Copyright IBM Corp. 2005. All rights reserved.

1.1 Tivoli Workload Scheduler overview

Tivoli Workload Scheduler is a multiplatform distributed scheduling system that provides high-volume, complex scheduling capability. A powerful scheduling language allows for precise coding of job streams with dependencies on other jobs, files, virtual resources, operator prompts, and jobs in other scheduling environments.

A Java™-based graphical user interface (GUI) known as the Job Scheduling Console (JSC) provides the user with the option of a dialog-based graphical interface. JSC enables a user to create and modify scheduling objects, to create an execution plan for a batch workload, and to monitor and manage the plan when it is executing. The more experienced user can use a command line interface (CLI) for monitoring, scheduling, and troubleshooting.

In this book, we explore the application programming interface (API) that enables TWS to be integrated with any application with a CLI or API. Chapter 8 of the *Tivoli Workload Scheduler 8.2 Reference Guide*, SC32-1274, provides an overview of the TWS extended agent API.

Job scheduling in TWS has two aspects: the database and the plan. The database contains all of the definitions for scheduling objects (for example, jobs, job streams, resources, and workstations). It also holds statistics about job and job stream execution, as well as information about the user ID that created an object and when an object was last modified. The plan contains all job scheduling activity planned for a one-day period. In TWS, the plan is created every 24 hours and consists of all jobs, job streams, dependencies, and other scheduling objects referenced in the upcoming day. All job streams for which you have created a run cycle are automatically scheduled and included in the plan. At the end of the day, the jobs and job streams not successfully executed can be rolled over into the next day's plan.

For more information about Tivoli Workload Scheduler V8.2, refer to *IBM Tivoli Workload Scheduler Version 8.2: New Features and Best Practices*, SG24-6628.

1.1.1 Software configurations used for this redbook

An IBM RS/6000® F50 workstation running AIX® V5.2 was used in the development of this redbook. The following software was loaded onto the system:

- IBM Tivoli Workload Scheduler V8.2
- IBM Tivoli Management Framework V4.1.1
- IBM Tivoli Job Scheduling Services V1.3
- IBM Tivoli Workload Scheduler Connector V8.2
- IBM Tivoli Storage Manager V5.3 for AIX

1.1.2 Tivoli Workload Scheduler concepts and terminology

TWS uses these important concepts:

- Job streams and calendars

  The job streams that are created using the JSC are central to the product's ability to manage batch job execution. Each job stream is scheduled to run on a specific set of dates and times, and consists of a list of jobs that execute as a unit (such as the weekly backup application), along with times, priorities, and other dependencies that determine the exact order of execution.

  Job streams are dated using actual dates, days of the week, or calendars. A calendar is a set of specific dates. You can create as many calendars as required to meet your scheduling needs. For example, you can define a calendar named PAYDAYS containing a list of pay dates, a calendar named MONTHEND containing a list of each last business day of the month for the current year, and a calendar named HOLIDAYS containing a list of your company's holidays. At the start of each processing day, TWS automatically selects all job streams that run on that day, and carries forward uncompleted job streams from the previous day.

- Workstations

  A workstation usually is an individual computer on which jobs and job streams are executed. A workstation definition is required for every computer that executes jobs in the TWS network. Workstation definitions primarily refer to physical workstations.

  However, in the case of extended agents and network agents, workstations are logical definitions that must be hosted by a physical TWS workstation.

  A TWS network has several types of workstations:

  - Master domain manager

    The domain manager in the topmost domain of a TWS network. It contains the centralized database files used to document scheduling objects. It creates the production plan at the start of each day, and performs all logging and reporting for the network.

  - Domain manager

    A domain manager acts as a repeater, which enables TWS to scale to large numbers of machines. The domain manager forwards messages between the master domain manager and the agents.

  - Backup master

    A Fault-tolerant Agent capable of temporarily assuming the responsibilities of its domain manager in a failover situation. This is enabled by marking the workstation as Full Status and Resolve Dependencies.

Archive

– Fault-tolerant Agent

  A workstation capable of resolving local dependencies and launching its jobs in the absence of a domain manager. The Fault-tolerant Agent is initialized at the beginning of the production day with all of the scheduling objects needed to perform the assigned workload.

– Standard agent

  A workstation that launches jobs only under the direction of its domain manager. Each job launch on the standard agent is directed in real time by the domain manager or master.

– Extended agent

  A logical workstation definition that enables you to launch and control jobs on other systems and applications, such as PeopleSoft, Oracle applications, SAP R/3, and MVS™ JES2 and JES3. Users can also write extended agents, as presented in this redbook.

– Network agent

  A logical workstation definition for creating dependencies between jobs and job streams in separate TWS networks.

1.1.3 Tivoli Workload Scheduler architecture

The master domain manager contains the centralized database files that are used to store scheduling objects. It creates the production plan at the start of each day, distributes the plan to the Fault-tolerant Agents and domain managers in the master domain, and logs all transactions on the TWS network.

All communications to agents are routed through the domain manager (the management hub in a domain). The network can be managed by a mix of agents. Fault-tolerant Agents can resolve local dependencies and launch jobs if a network interruption causes them to lose communication with their domain managers, because each one has a copy of the current plan (scheduling instructions for a given day) at the beginning of every processing day.

1.1.4 Extended agents

TWS extended agents are programs that enable TWS to manipulate and get information about objects in other application environments. Extended agents use open scheduling APIs and protocols, and they provide an effective mechanism for extending TWS scheduling capabilities to foreign platforms and

Archived

applications. Extended agents also allow segregation of applications at the workstation level by providing an alternative method to the *jobmanrc* process. This enables some applications to be treated differently using an alternative job scheduling method.

Figure 1-1 shows the connection between TWS and an extended agent. The extended agent is installed on one of the TWS hosts. TWS accepts information from the extended agent through an interface called the *access method*, which is a shell script or program that resides in the ~*TWSHOME*/methods directory.

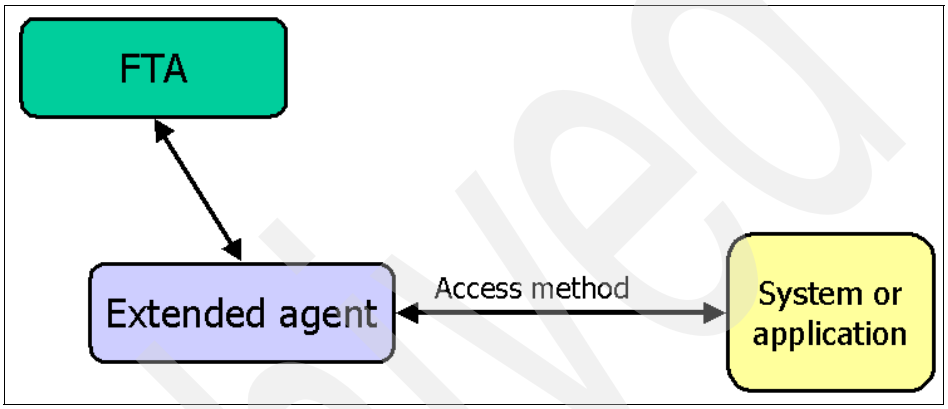

Figure 1-1 Extended agent network

Figure 1-2 on page 6 explains extended agent processing. The sequence of operations follows.

Note: This explanation does not consider network communication issues. Even though the extended agent must reside on a Fault-tolerant Agent, it may interface with APIs on other hosts. This is commonly done via socket connections or application clients.

1. Batchman process on the Fault-tolerant Agent talks to the jobman (jobman.exe on a Microsoft® Windows® system), which is owned by the TWS install user ID on Windows.
2. Jobman invokes JOBMAN (jobmon.exe on an Windows system) process in the context of the TWS user (maestro, for example).
3. JOBMAN talks to the access method.
4. The access method invokes a Remote Function Call (RFC).
5. The access method consults with the <method.opts> file.
6. The access method talks to the system or the application's job.

Archived

7. The access method and the job communicate with each other.
8. The method passes the information back to the TWS host through JOBMAN.

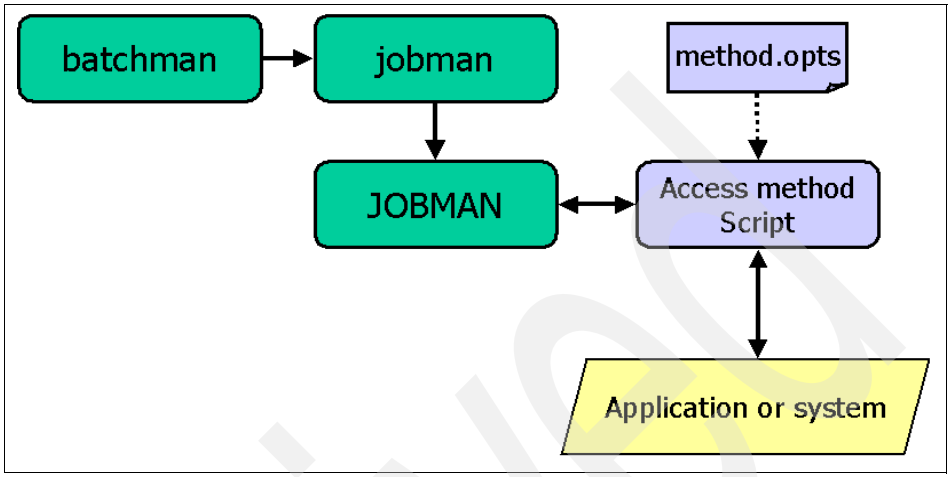

Figure 1-2 Extended agent processing

The JOBMAN process launches the access method script to perform one of these tasks:

- Launch a job.
- Manage a job.
- Check for the existence of a file to satisfy an OPENS dependency.
- Get the status of an external job.

The syntax of the method execution is:

```
methodname -t task options -- taskstring
```

In this syntax:

- *task* can be:

  LJ Launch a job
  MJ Manage a previously launched job
  CF Check availability of a file – OPENS dependency
  GS Get the status of an external job

- *options* is the list of job properties.
- *taskstring* is the string to execute from the job definition.

Archive

These are the options (related to the job's properties) that should be passed to the access method:

- Workstation/Host/Master
- Workstation's node definition
- Workstation's port definition
- The current production run number
- The job's run number
- The job's schedule name
- The date of the schedule (in two formats)
- The user ID (logon) for the job
- The path to the output (stdlist) file for the job
- The job ID
- The job number
- The string from the SCRIPTNAME or DOCOMMAND entry in the job definition

Example 1-1 shows a method invocation in which the job `TEST` is executed on the extended agent workstation `ITSO6`, using the user ID `itso` and method name `wgetmethod`.

Example 1-1 Example method invocation

```
wgetmethod -t LJ
-c ITSO6,ITSO7,ITSO7
-n ITSO6
-p 31111 -r 143,143 -s MACDSH91
-d 20000410,955381986 -l itso
-o /opt/maestro/stdlist/2000.04.10/013676.1053
-j TEST,13676 -- /home/itso//batchjobs/killer
```

TWS currently provides these extended agents:

- UNIX® Remote Shell
- UNIX Local
- MVS (JES,OPC, CA7)
- SAP R/3 Batch
- PeopleSoft
- Oracle Applications

UNIX Remote Shell and UNIX Local extended agents come with the TWS base package. Other extended agents are bundled into a separate package called Tivoli Workload Scheduler for Applications.

Archived

Note: In addition to the TWS-provided extended agents, you can write your own extended agents for platforms or applications that are not supported by TWS by using the open API documented in the *IBM Tivoli Workload Scheduler 8.2 Reference Guide*, SC32-1274.

1.2 Tivoli Storage Manager overview

TSM is an end-to-end, scalable storage management solution spanning handhelds to mainframes on more than 35 platforms. Features include:

- Centralized storage management
- Storage Area Network (SAN) features, such as LAN-free backup and tape sharing
- Automated network incremental and subfile backup, archive, and retrieval
- Fast recovery time
- Space management file migration
- High-speed policy-based disaster recovery
- Data protection offered for most popular groupware, e-mail, databases, and applications

TSM is the core product of the Tivoli Storage Management product set. It provides a solution for distributed data and storage management in an enterprise network environment. It is the next generation of the product originally released by IBM as ADSTAR Distributed Storage Manager (ADSM).

TSM and its complementary products provide these base functions:

- Data protection, including:
  - Operational backup and restoration of data: The backup process creates a copy of the data to protect against the operational loss or destruction of file or application data. The customer defines how often to back up (frequency) and how many numbers of copies (versions) to hold.

    The restore process places the backup copy of the data into a customer-designated system or workstation.
  - Disaster recovery: All activities to organize, manage, and automate the recovery process from a major loss of IT infrastructure and data across the enterprise. This includes processes to move data off-site into a secure vault location, to rebuild IT infrastructure, and to reload data successfully in an acceptable time frame.

Archived

- Storage resource management, including:
  - Vital record retention, archive, and retrieval: The archive process creates a copy of a file or a set of files representing an endpoint of a process for long-term storage. Files can remain on the local storage media or can be deleted. The customer controls the retention period (how long an archive copy is to be retained).

    The retrieval process locates the copies within the archival storage and places them into a customer-designated system or workstation.
  - Hierarchical space management: This process provides the automatic and transparent movement of operational data from the user system disk space to a central storage repository. If the user accesses this data, it is dynamically and transparently restored to client storage.

1.2.1 What are the benefits of a TSM Extended Agent?

TSM administrators must perform several types of operations regularly each day. TSM contains a built-in scheduling facility, which provides a simple mechanism to automate routine tasks. By extending TWS to schedule these tasks you can:

- Create job dependencies.
- Set limits.
- Set resources.
- Set workstation classes (so that a single job definition can be used by a list of workstations).

This scheduling facility does not provide the ability to assign dependencies among scheduled tasks or to assign limits or priorities. By extending TWS to schedule these tasks, you can take advantage of its advanced scheduling capabilities.

The following common TSM tasks were implemented for this example:

- Database backup (BACKUP DB)
- Volume history backup (BACKUP VOLHISTORY)
- Device configuration backup (BACKUP DEVCONFIG)
- Delete volume history (DELETE VOLHISTORY)
- Inventory expiration (EXPIRE INVENTORY)
- Client backup

All TSM tasks executed by the TSM Extended Agent (Tivoli Storage Managerxagent), use the Tivoli Storage Manager Administrative Client Interface (**dsmadmc**). The user ID used to log on to the interface is fetched from the Tivoli Storage ManagerAdmin variable in the Tivoli Storage Managerxagent.opts file in the Tivoli Workload Scheduler methods directory. The password is retrieved from

Archive

the Tivoli Workload Scheduler parameter object with the same name as the user ID, so different passwords can be set for different user IDs. In turn, TWS extended agent can be deployed on multiple TSM servers with unique user IDs.

As shown in Figure 1-3, extended agents are used to extend TWS job scheduling functions to other systems and applications.

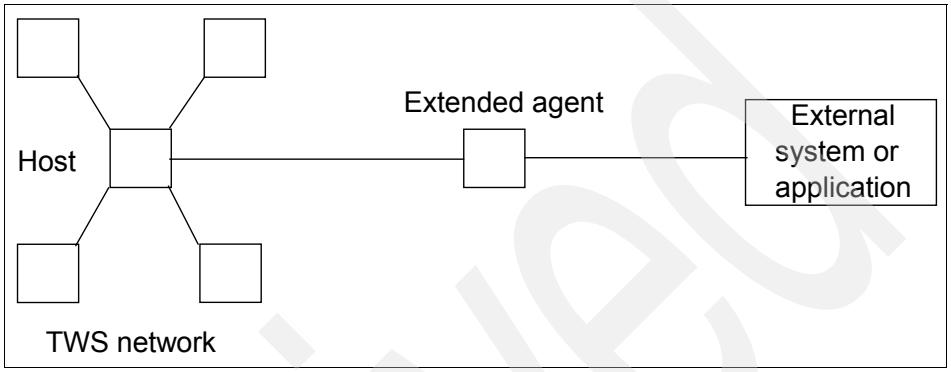

Figure 1-3 Extended agent

An extended agent is defined as a workstation that has a host and an access method. The host is any other workstation, except another extended agent. The access method is a Tivoli-supplied or user-supplied script or program that the host executes whenever the extended agent is referenced in the production plan. For example, to launch 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 or application to launch the job and return the status of the job.

Each extended agent must have a logical workstation definition. This logical workstation must be hosted by a TWS physical workstation (a master, domain manager, or Fault-tolerant Agent workstation). The extended agent workstation definition references the name of the access method and the host workstation. When jobs are launched on the extended agent workstation, the access method is called and passes the job information to the external system. For an example of defining an extended agent workstation, see the *IBM Tivoli Workload Scheduler 8.2 Planning and Installation Guide,* SC32-1273.

1.3 Summary

Tivoli Workload Scheduler is a multiplatform distributed scheduling program that provides high-volume, complex scheduling capability for many systems and applications. Master domain manager, domain manager, backup master,

Archived

Fault-tolerant Agent and standard agent are various types of workstations in a Tivoli Workload Scheduler network.

You can extend the number of platforms and applications that TWS supports by writing TWS extended agents, which enable TWS to manipulate and get information about objects in other scheduling environments.

Tivoli Storage Manager is a scalable storage management solution spanning handhelds to mainframes on more than 35 platforms. Centralized storage management, LAN-free backup and tape sharing, and automated network incremental and subfile backup, archive, and retrieval are some of the important functions that TSM provides.

Archived

Archived

2

Extended agent functions

This chapter describes the functions of the Tivoli Workload Scheduler extended agent method, the access method interface, and the communications protocols between jobs running within an access method.

This chapter has the following sections:

Archived

© Copyright IBM Corp. 2005. All rights reserved.

2.1 Introduction

The extended agent is defined as a workstation that has a host and an access method. The host is any other TWS workstation, except another extended agent. The access method is a Tivoli-supplied or user-supplied script or program that the host executes whenever the extended agent is referenced in the production plan.

For example, to launch 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 or application to launch the job and return the status of the job.

2.2 Workstation definition

Each extended agent must have a logical workstation definition. This logical workstation must be hosted by a TWS physical workstation, either a master, domain manager, or Fault-tolerant Agent workstation. The extended agent workstation definition references the name of the access method and the host workstation. Workstation definitions are created using the command-line Composer interface, or Job Scheduling Console (JSC). When jobs are launched on the extended agent workstation, the access method is called and passes the job information to the external system.

Suppose, for example, that you have a TWS network comprised of three workstations: a master workstation (called Copernicus), a Fault-tolerant Agent workstation (Kepler), and an extended agent workstation (Galileo). This is an example of the extended agent workstation definition:

```
CPUNAME galileo
OS OTHER
NODE focus
TCPADDR 1609
FOR MAESTRO
   HOST kepler
   ACCESS nuncius
END
```

In this case, Galileo is the extended agent workstation, `focus` is the node name, `1609` is the TCP port, Kepler is the host, and `nuncius` is the name of the access method. An executable file (script or program) named nuncius must reside in the /methods subdirectory of the TWS installation and must be executable. For extended agents only, the node and tcpaddr definitions are left to the method developer to define. In other words, depending on the nature of the access method, node might have different uses.

Archived

2.3 Method options file

By default, when executing an extended agent job, TWS tries to execute the method file as the logon defined for each job to be run by the method. When executing the method script to check for an OPENS dependency (using the check file, or CF, task) TWS executes the method as root by default. You can change this behavior by using a method options file to specify special login information and other options for your access method. For example, you may wish to use the logon field of your agent's job definitions to represent a logon to a foreign system that does not exist on the extended agent's host.

TWS reads the file, if it exists, before executing a method. If the file is modified after TWS is started, the changes take effect when it is stopped and restarted. The file can contain TWS options and any other method information you wish to include. The options recognized by TWS are:

```
LJuser=username
CFuser=username
```

`LJuser=username` specifies the logon to use for the launch job (LJ) and manage job (MJ) tasks. The default is the logon from the job definition.

`CFuser=username` specifies the logon to use for the CF task. The default is root for UNIX, and for Windows it is the user name of the account in which TWS was installed.

Note: If the extended agent's host is a Windows computer, these users must be defined as TWS user objects. The options file must have the same path name as its access method, with an .opts file extension. For example, the Windows path name of an options file for a method named nuncius is WShome\methods\nuncius.opts.

Archive

2.4 Access method interface

The interface between TWS and an access method consists of information passed to the method on the command line, and messages returned to TWS in stdout.

2.4.1 Method command line syntax

The TWS host runs an access method using the following command line syntax:

```
methodname -t task options -- taskstring
```

methodname specifies the file name of the access method. In our example, this is nuncius. All access methods must be stored in the WShome/methods directory.

-t task specifies the task to be performed, where *task* is one of the options listed in Table 2-1.

Table 2-1 Task types

Task	Description
LJ	Launches a job. The access method is called with this task option when TWS launches a job. TWS will expect the access method to continue running until the job it executes is complete.
MJ	Manages a previously launched job. Use this option to synchronize job(s) if a prior LJ task terminated unexpectedly. The MJ method is called for under only a few circumstances. For example, TWS or its batchman process terminates (an operator issues a Conman STOP command to the agent's host) while an extended agent job is running. When the TWS processing is restarted, it will execute the extended agent's access method for each job that was previously running, passing it the MJ task and the same arguments initially sent to the job. The access method should use the MJ task to determine the state of the previously running job and report that state back to TWS.
CF	Checks the availability of a file. Use this option to check file OPENS dependencies. Using the Galileo workstation example, if a schedule contained the dependency (OPENS GALILEO#/this/is/my/file) TWS would call the access method for Galileo and pass it the CF task and the path (/this/is/my/file).

options specifies the options associated with the task. See 2.4.2, "Task options" on page 17 for more information.

taskstring is a string of up to 255 characters associated with the task, as explained in the next section, "Task options."

Archived

2.4.2 Task options

The task options are listed in Table 2-2. An “x” means the option is valid for the task.

Table 2-2 Task options

Task	Options										Task string
-t	-c	-n	-p	-r	-s	-d	-l	-o	-j	-q	
LJ	x	x	x	x	x	x	x	x	x		ljstring
MJ	x	x	x	x	x	x	x	x	x		mjstring
CF	x	x	x							x	cfstring

-c *xagent,host,master* Specifies the TWS names of the extended agent, the host, and the master domain manager, separated by commas.

-n *nodename* Specifies the node name of the computer associated with the extended agent, if any. This is defined in the extended agent’s workstation definition node field.

-p *portnumber* Specifies the TCP port number associated with the extended agent, if any. This is defined in the extended agent’s workstation definition TCP Address field.

-r *currentrun,specificrun* Specifies the current run number of TWS and the specific run number associated with the job, separated by a comma. The current and specific run numbers might be different if the job is carried forward from an earlier run.

-s *jstream* Specifies the name of the job’s job stream.

-d *scheddate,epoch* Specifies the schedule date (yymmdd) and the epoch equivalent, separated by a comma.

-l *user* Specifies the job’s user name. This is defined in the job definition Logon field.

-o *stdlist* Specifies the full path name of the job’s standard list file. Any output from the job must be written to this file.

-j *jobname,id* Specifies the job’s name and the unique identifier assigned by TWS, separated by a comma. The name is defined in the job definition Job Name field.

-q *qualifier* Specifies the qualifier to be used in a test command issued by the method against the file.

Archive

-- ljstring Used with the LJ task. The string from the script file or command field of the job definition.

-- mjstring Used with the MJ task. The information provided to TWS by the method in a %CJ response to an LJ task. Usually, this identifies the job that was launched. For example, a UNIX method can provide the process identification (PID) of the job it launched, which TWS sends as part of an MJ task.

-- cfstring Used with the CF task. For a file OPENS dependency, the string from the Opens Files field of the job stream definition.

2.4.3 Example

Using our previous workstation definition and the options we just listed, here is an example of how your access method would be called from TWS. In this example, a job defined as `tsmjob1` runs the command `ADMIN VHBACKUP /opt/tsm/dat/fa20001020` in the job stream (schedule) `DAILYBAK`:

```
/opt/maestro/methods/nuncius -t LJ -c GALILEO,KEPLER,COPERNICUS -n focus -p
1609 -r 109,109 -s DAILYBAK -d 20001207,976147200 -l root -o
/opt/maestro/stdlist/2000.12.07/09041.1414 -j TSMJOB1,9041-- ADMIN VHBACKUP
/opt/tsm/dat/fa20001020
```

Examining the command structure from the example:

`/opt/maestro/methods/nuncius` This is the directory to the access method script.

`-t LJ` The task option LJ was sent, indicating that a job is being run.

`-c GALILEO,KEPLER,COPERNICUS` This option shows that Galileo is the extended agent's name, Kepler is the agent's host's name, and Copernicus is the TWS master's name.

`-n focus` Focus is the node name of the extended agent. This name comes directly from the workstation definition for Galileo.

`-p 1609` The number 1609 is the tcpaddr port definition from the workstation definition for Galileo.

`-r 109,109` The current run number (Tivoli Workload Scheduler production day) is 109, as is the job's run number. If, for example, this option was -r 110,109, it would signify that the job stream (schedule) from which this job is running has been carried forward from a previous day.

`-s DAILYBAK` The job stream (schedule) name from which this job is running is DAILYBAK.

`-d 20001207,976147200`
The current scheduling production run date is December 7, 2000. Remember that the scheduling production run date may be different than the actual calendar run date.

`-l root` The job run by this method should log on to the system as root. This information comes from the job definition.

`-o /opt/maestro/stdlist/2000.12.07/O9041.1414`
All output (stdout and stderr) is being redirected to the file /opt/maestro/stdlist/2000.12.07/O9041.1414 on the host workstation.

`-j TSMJOB1,9041`
The name of the job being run is TSMJOB1, and its job number (as seen in the Conman SHOWJOBS command) is 9041. On UNIX systems only, 9041 is also the parent process ID for this job.

The rest of the arguments after a double hyphen are directly from the SCRIPTNAME or DOCOMMAND entry in the TWS job definition. In this case, the application understands the command ADMIN VHBACKUP and processes the command appropriately:

```
-- ADMIN VHBACKUP /opt/tsm/dat/fa20001020
```

Typically, the access method ignores any options after the double hyphen and passes them directly to the target application. This is, of course, up to the method's author to determine.

Example 2-1 shows the job's status in the Console Manager upon execution.

Example 2-1 Job status

```
Est)    (Est)
CPU      Schedule   Job    State Pr Start  Elapse  Dependencies

GALILEO #DAILYBAK ******** SUCC  10 14:14  01:10
                  TSMJOB1  SUCC  10 14:14  01:10  #J9041
%
```

Archive

2.5 Method response messages

Methods returns information to TWS in messages written to stdout. Each line starting with a percent sign (%) and ending with a new line is interpreted as a message. The messages have the following format:

```
%CJ state [mjstring]
%JS [cputime]
%UT [errormessage]
```

The elements of this format are:

CJ	Changes the job state.
state	The state to which the job is changed. All TWS job states are valid except hold and ready.
mjstring	A string of up to 255 characters that TWS will include in any MJ task associated with the job.
JS [*cputime*]	Indicates successful completion of a job and provides its elapsed run time in seconds. For the job to be considered successful, this message must be presented (usually with an echo statement) before the method ends.
UT [*errormessage*]	Indicates that the requested task is not supported by the method. Displays a string of up to 255 characters that TWS will include in its error message (stored in the Tivoli Workload Scheduler standard list file; typically this file is ~maestro/stdlist/yyyy.mm.dd/MAESTRO).

You can use these messages within your access method to relay information to the Tivoli Workload Scheduler Console Manager. By changing the status of your job, operators can pay special attention to your jobs by filtering their console windows by job status. For example, you might have jobs toggle between wait and exec states by issuing an **`echo %CJ WAIT`** or **`echo %CJ EXEC`** command while the job connects to the application and executes its function.

Archived

2.6 Execution and troubleshooting

The extended agent's access method is executed by the TWS jobman process, much as a standard job is launched through jobmanrc.

2.6.1 Executing the method

Before the method is launched, TWS establishes an execution environment for the method.

The LJuser parameter is read from the method options file to determine the user account with which to run the method. If the parameter is not present or the options file does not exist, the user account specified in the Logon field of the job's definition is used. In addition, the following environment variables are set:

HOME	User's home directory
LOGNAME	Login user's name (from the job definition)
PATH	For UNIX, /bin:/usr/bin; for Windows, %SYSTEM%\SYSTEM32
TZ	Time zone

2.6.2 Killing a job

While an access method is running an MJ or LJ task, it should trap a SIGTERM signal (signal 15). This is the signal sent to the job when an operator issues a kill command from the user interface. When a kill signal is caught, the method should attempt to stop or kill the job and exit without writing a %JS message.

2.6.3 Method troubleshooting

If the method cannot be executed, its state is set to `fail`. All output messages from an access method, except those that start with a percent sign (%), are written to the job's standard list (stdlist) file. For GS and CF tasks that are not associated with TWS jobs, messages are written to the TWS standard list file. For information regarding a problem of any kind, check these files.

For extended agents, error, warning, and information messages are written to the TWS stdlist file.

- A successful job launch generates the following message:

  `Launched job` *`jobname`* `for` *`wkstation`* `,#J` *`jobid`* `for user` *`username`*

- Failure to launch a job generates the following message:

  `Error launching` *`jobname`* `for` *`wkstation`* `:`*`errortext`*

Archived

- Failure of a CF task generates the following message:

  ```
  Error invoking methodname for wkstation :errortext
  ```

- Failure of an MJ task generates the following message:

  ```
  Error managing jobname for wkstation using methodname :errortext
  ```

- When a method sends a message to Jobman that is not recognized, the following message is generated:

  ```
  Error:message invalid for jobname ,#j jobnumber for wkstation using
  methodname ."first 64 characters of the offending message"
  ```

2.7 Summary

In this chapter we covered the functions of the TWS extended agent method, the Access Method Interface, and the communications protocols between jobs running within an access method.

The main functions used in an extended agent method are:

LJ Launch job
MJ Manage job
CF Check file dependency

The extended agent's access method is executed by the jobman process, much as a standard job is launched through jobmanrc.

Archived

3

Case study: TSM Extended Agent

This chapter includes scripts that are useful to Tivoli Workload Scheduler and Tivoli Storage Manager administrators.

These scripts are about TSM processes such as backup, storage pools, migration, reclamation, volume history, and device configuration. TSM administrators generally schedule these processes, so it makes sense to write a Tivoli Storage Manager Extended Agent for Tivoli Workload Scheduler (*TSM Extended Agent*) that incorporates these functions. Using the TSM Extended Agent, administrators can schedule these TSM operations through TWS.

This chapter has the following sections:

© Copyright IBM Corp. 2005. All rights reserved.

3.1 Tivoli Storage Manager operations

We start by providing some background information about these operations. The following sections should be especially useful if you are not familiar with TSM scheduling operations.

- Back up database

  Use this command to back up a TSM database to sequential access volumes (Figure 3-1). To determine how much additional storage space a backup will require, use the QUERY DB command. This command displays the database pages (in megabytes) that have changed since the last backup.

  The syntax of the command is:

  ```
  backup db devclass=dbbackup type=full scratch=yes wait=yes,
  ```

  In this syntax:

  backupdb — The name of the devclass defined on the TSM server.

  type — Specifies that you can run the full backup on the TSM database. This parameter is optional. Possible options are: incremental, full, and dbsnapshot. The default is *incremental*.

  scratch — To take the device that is scratch. The default value is *yes*. Using scratch=no means that the scratch volume cannot be used. This parameter is optional.

  wait — Specifies whether to wait for the server to complete processing this command in the foreground. The default is *no*. Possible options are: yes (the server processes the command in foreground) and no (the server processes the command in background).

  If a backup completes unsuccessfully, you can verify and restart it.

  You can find more details about the syntax of the commands in the *IBM Tivoli Storage Manager for AIX Administrator's Reference V5.3*, GC32-0769.

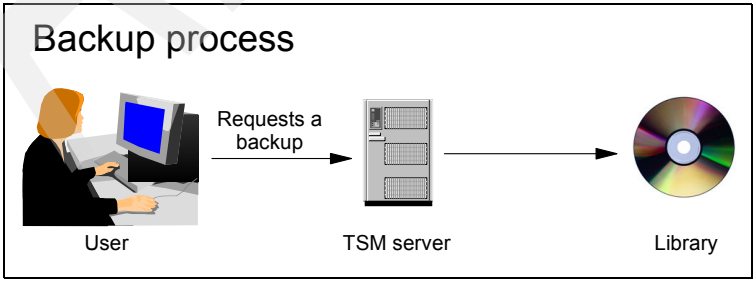

Figure 3-1 Requesting a backup

Archived

- Back up storage pool

  Use this to back up primary storage pool files to a copy storage pool.

  If a file already exists in the copy storage pool, the file is not backed up unless the copy is marked as damaged. However, a new copy is not created if the file in the primary storage pool is also marked as damaged. In a random-access storage pool, neither cached copies of migrated files nor damaged files are backed up.

  If migration for a storage pool starts during a storage pool backup, some files may be migrated before they are backed up. You may want to back up storage pools that are higher in the migration hierarchy before backing up storage pools that are lower. For example, when performing a storage pool backup to copy the contents of a storage pool off-site, if the process is not done according to the existing storage pool hierarchy, some files may not be copied to the copy storage pool. This could be critical for disaster recovery purposes. When performing a storage pool backup on multiple storage pools, the primary storage pool should be completed before the backup process on the next storage pool.

  The syntax of the command is:

  ```
  backup stgpool spacemgtpool copypool
  ```

  In this syntax:

spacemgtpool	The primary backup that you want to back up.
copypool	The copy storage pool defined on the TSM server.

- Reclamation

  Specifies when the server reclaims a volume, based on the percentage of reclaimable space on a volume (Figure 3-2 on page 26). Reclamation makes the fragmented space on volumes usable again by moving any remaining active files from one volume to another volume, thus making the original volume available for reuse. You can specify an integer from 1 to 100. The value 100 means that reclamation is not performed.

  If you change the value from the default of 100, specify a value of 50 percent or greater so that files stored on two volumes can be combined into a single output volume.

  To move data from the reclaim storage pool back to the original storage pool, use the storage pool hierarchy. Specify the original storage pool as the next storage pool for the reclaim storage pool.

  The syntax of the command is:

  ```
  update stgpool copypool reclaim=50
  ```

  In this syntax, `reclaim` is the percentage of the reclamation in copypool.

Archive

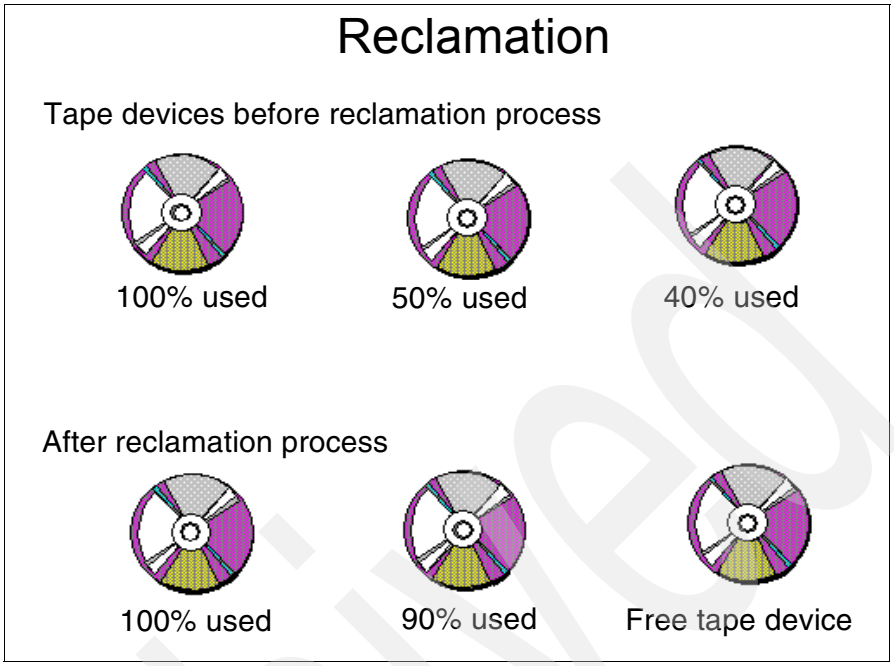

Figure 3-2 Reclamation in the tape device

- Migration

  You can use a primary storage pool as the destination for backup files, archive files, or files migrated from client nodes (Example 3-1).

Example 3-1 Migration command

```
define stgpool somepool dbbackup pooltype=primary access=readwrite
maxsize=nolimt highmigh=90 lowmig=70 migprocess=1 collocate=no migcontinue=yes
reusedelay=0 migdelay=0
```

In this syntax:

somepool	The pool name defined on the TSM server.
dbbackup	The device class name defined on the TSM server.
pooltype	Specifies that you want to define a primary storage pool. The default value is *primary*.
access	Specifies how client nodes and server processes (such as migration and reclamation) can access files in the storage pool. The default value is *readwrite*, but you can also define as readonly and unavailable.

Archived

maxsize Specifies the maximum size for a physical file size that the server can store in the storage pool during a session with a client. The default value is *nolimit*.

highmig Specifies that the server starts migration for this storage pool when the amount of data in the pool reaches this percentage of the pool's estimated capacity. You can specify an integer from zero to 100. The default value is *90*. When the storage pool exceeds the high migration threshold, the server can start migration of files by node, to the next storage pool, as defined with the NEXTSTGPOOL parameter. You can specify highmig=100 to prevent migration for this storage pool.

lowmig Specifies that the server stops migration for this storage pool when the amount of data in the pool reaches this percentage of the pool's estimated capacity. You can specify an integer from zero to 99. The default value is *70*. When the storage pool reaches the low migration threshold, the server does not start migration of another node's files. Because all file spaces that belong to a node are migrated together, the occupancy of the storage pool can fall below the value you specified for this parameter. You can set lowmig=0 to permit migration to empty the storage pool.

migprocess Specifies the number of processes the server uses for migrating files from this storage pool. You can specify an integer from 1 to 999. The default value is *1*. During the migration the server runs this number of processes in parallel to provide the potential for improved migration rates.

migdelay Specifies the minimum number of days a file must remain in a storage pool before the file becomes eligible for migration. The server counts the number of days from the day the file was stored in the storage pool or retrieved by a client, whichever is more recent. The default is zero, which means you do not want to delay migration.

migcontinue Specifies whether you allow the server to migrate files that do not satisfy the migration delay time. The default value is *yes*. Because you can require that files remain in the storage pool for a minimum number of days, the server may migrate all eligible files to the next storage

pool yet not meet the low migration threshold. With this parameter, you can specify whether the server is allowed to continue the migration process by migrating the files that do not satisfy the migration delay time.

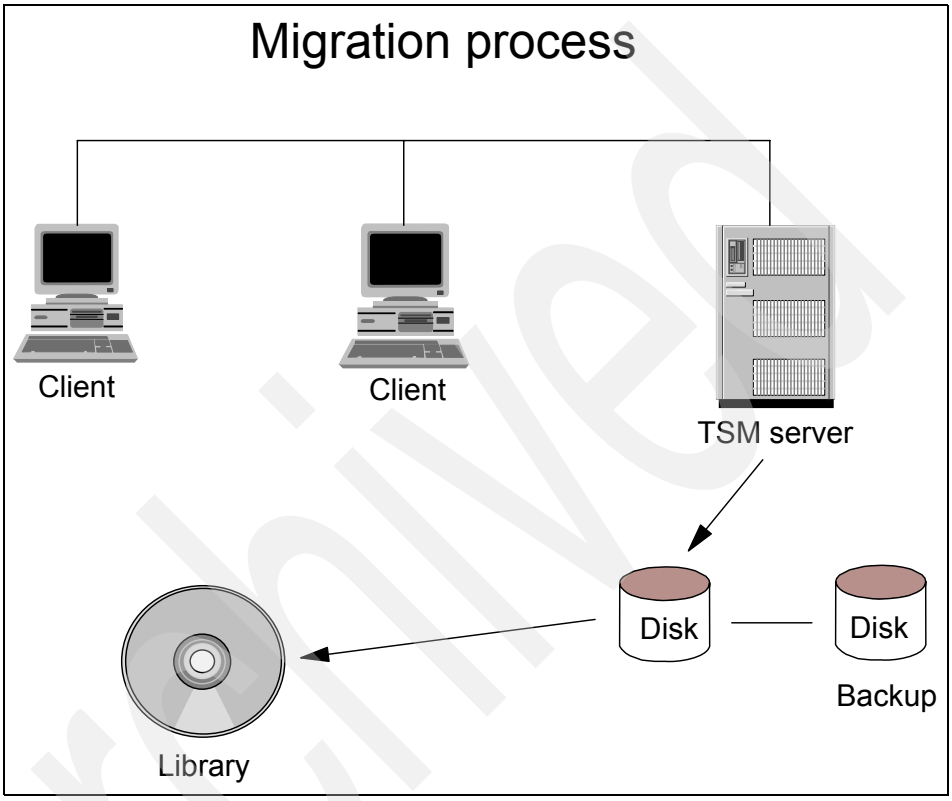

Figure 3-3 Migration process

- Back up device configuration

  Use this command to back up the following information in one or more files:

  - Device class definitions
  - Library definitions
  - Drive definitions

  To restore the TSM database, the device configurations must be available. You can use the devconfig server option to specify one or more files for storing device configuration information. TSM updates the files whenever a device class, library, or drive is defined, updated, or deleted.

Archived

The syntax of the command is:

```
backup devconfig filenames=device
```

In this syntax, *device* is the directory into which devconfig information will be backed up.

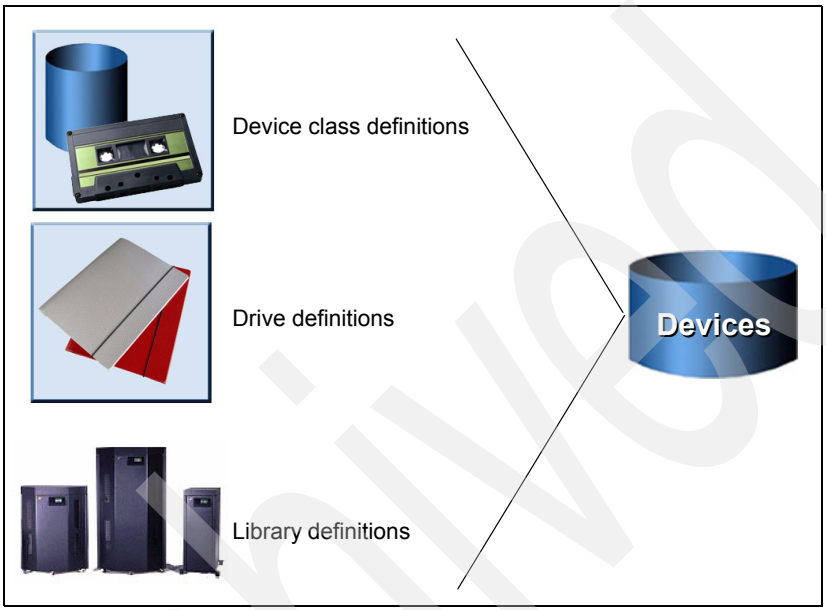

Figure 3-4 Device configuration

- Back up volume history

  Use this command to back up sequential volume history information to one or more files. You can use volume history information when you reload the database and audit affected storage pool volumes. If you cannot start the server, you can use the volume history file to query the database about these volumes. The volume history includes information about the following types of volumes:

  - Database backup volumes
  - Database dump volumes
  - Export volumes
  - The following sequential access storage pool volumes:
    - Volumes added to storage pools
    - Volumes reused through reclamation or move-data operations

Archived

- Volumes removed by using the delete volume command or during reclamation of scratch volumes

The syntax of the command is:

```
backup volhistory filenames=volhist
```

volhist is the name of the filename that will contain information about the volhistory backup.

- Delete volume history

Use this command to delete volume history file records that are no longer needed (for example, records for obsolete database backup volumes).

When you delete records for volumes that are not in storage pools (such as database backup or export volumes), the volumes return to Source Manager, which acquires them as scratch volumes. Scratch volumes of device-type files are deleted. When you delete the records for storage pools, volumes remain in the TSM database. When you delete records for recovery-plan file objects from a source server, the objects on the target server are marked for deletion.

The syntax of the command is:

```
delete volhistory type=rpfile todate=11/31/02
```

type Specifies the type of records, which also meet the date and their criteria, to delete from the volume history file. *rpfile* is for deleting only records that contain information about full and incremental database backup volumes and recovery plan file volumes.

todate Specifies the date to use to select sequential volume history information to be deleted. TSM deletes only those records with a date on or before the date you specify.

For more details, refer to the *IBM Tivoli Storage Manager for AIX Administrator's Reference V5.3*, GC32-0769.

- Expire inventory

Use this command to manually start inventory expiration processing. This inventory expiration process removes the client backup and archive file copies from server storage based on the policy specified in the backup and archive copy groups of the management classes to which the files are bound. The inventory expiration process that runs during server initialization does not remove these virtual volumes.

Only one expiration process is allowed at any time. If an expiration process is running you can not start another process.

The syntax of the command is:

```
expire inventory quiet=no wait=no skipdirs=no duration=40
```

Archive

In this syntax:

quiet Specifies whether the server suppresses detailed messages about the policy changes during the expiration processing. This parameter is optional. The default value is *no*, which specifies that the server sends detailed informational messages. You can also set the value to yes, which specifies that the server sends only summary messages.

wait Specifies whether to wait for the server to complete processing this command in the foreground. Possible values are no, which specifies that the server processes this command in the background, and yes, which specifies that the server processes this command in the foreground. This parameter is optional.

skipdirs Specifies whether the server skips directory type during the expiration processing. Possible values are: no, which specifies that the server will expire files and directories based on the appropriate policy criteria, and yes, which specifies that the server will skip directory type objects during expiration processing even if the directories are eligible for expiration.

duration Specifies the maximum number of minutes for the expiration process to run. The process stops when the specified number of minutes have passed or when all eligible expired objects are deleted, whichever comes first. You can specify a number from 1 to 999,999 (optional).

- Restore database

  To restore a database or volume to its most current state, the log mode must have been set to roll forward continuously from the time that the last backup series was created.

  If the log mode is set to roll forward after a point-in-time database restoration, a database backup starts when the server is brought up for the first time. This can cause loss of data: A tape can have current data on it, but because of the point-in-time restoration, it can be marked as scratch. When the server starts for the first time, TSM may use this tape to write the database backup, thus destroying the original data on this tape.

  You can restore a database if the following are true:

  – The log mode was set to roll-forward continuously from the time the last backup series was created.
  – An intact recovery log is available.
  – An intact volume history file is available.

TSM requests volume mounts to load the most recent backup series, then uses the recovery log to update the databases to its most current state.

If the volume history file is unavailable, you can use one or more **dsmc restore db** commands to restore the database to a specific point in time. For example, to load a full backup and one or more incremental backups, issue a **dsmc restore db** command for the full backup and an additional **dsmc restore db** command for each incremental backup. When you use multiple **dsmc restore db** commands, specify **commit=no** for each command except the last one. For the last command, specify **commit=yes**. The database remains in an inconsistent and unusable state until you issue a **dsmc restore db** command with a **commit=yes.**

The syntax of the command is:

```
dsmc restore db devclass=device_class_name volumename=volume_name
commit=no
```

In this syntax:

devclass Specifies the name of the sequential access device class to use. The device class must be defined in a device configuration file.

volumename Specifies the backup volume to use to restore the database. Possible values are:

- *volume_name* Specifies the names of the volumes. To specify multiple volumes, separate the names with commas without intervening spaces.
- *file:file_name* Specifies the name of a file that contains a list of the volumes.

commit Specifies whether this is the last restore command needed to restore the database. The default value is *no* (specifies that you will issue one or more additional **dsmc restore db** commands), but you can also define the value yes (specifies that this is the last restore command to restore the database). This parameter is optional.

Archived

3.2 Testing the TSM Extended Agent solution

After you analyze all of the scripts, you can also test the environment and commands in the TWS structure.

Note: Appendix A, “TSM Extended Agent source code” on page 79 includes the source code for the TSM Extended Agent script. You can also download it from the ITSO Web site. For downloading instructions, refer to Appendix B, “Additional material” on page 101. Note that the TSM Extended Agent script runs in Korn Shell environment. We have tested the script on AIX, but it should run with little or no modification on any platform that supports Korn Shell.

Decide which system will host your master (the system on which the TWS database resides, as well as the system on which all updates are received) and install the TWS software on your system. Follow the installation steps in the *IBM Tivoli Workload Scheduler 8.2 Planning and Installation Guide,* SC32-1273 and *IBM Tivoli Workload Scheduler Job Scheduling Console 1.3 User’s Guide,* SC32-1257.

To access the TWS database, start the Job Scheduling Console (JSC). Log in as user `twsuser` (default TWS user), or the name of the TWSuser you have chosen.

Figure 3-5 on page 34 shows the welcome window, which is presented when logging on to the JSC as a first-time user. You may either select the radio button for **Dismiss this window and work on my own** and click **OK**, or simply click **Cancel** to dismiss this window. If you do not want to see this window the next time you log on to the JSC, select **Don’t show this window again**, select **Dismiss this window and work on my own**, and click **OK**.

Archive

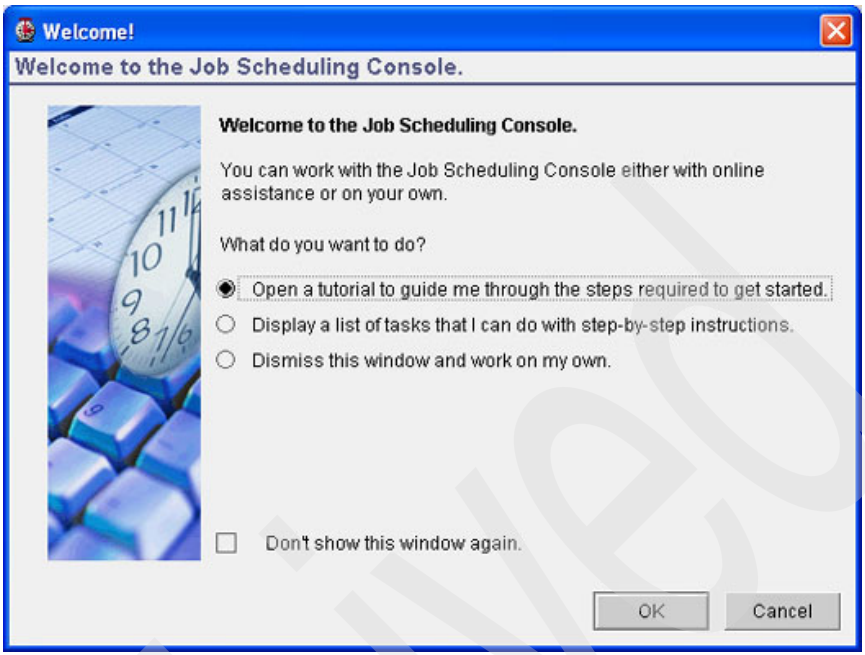

Figure 3-5 JSC welcome window

The JSC main window (Figure 3-6 on page 35) presents the Action list pane and the Work with engines pane, which provide users with access to the TWS databases and the current plan.

If you close the Action list pane, you can recall it by moving your cursor over the folder tab on the far left side of the JSC and pushing the pin in the upper-right corner of its pane, or by pressing Ctrl+T, or by selecting **View** → **Show** → **Portfolio** from the JSC menu bar. To re-open the Work with engines pane, click the button on the tool bar of the JSC main window. The functionality of these windows is explained later in this document.

Archive

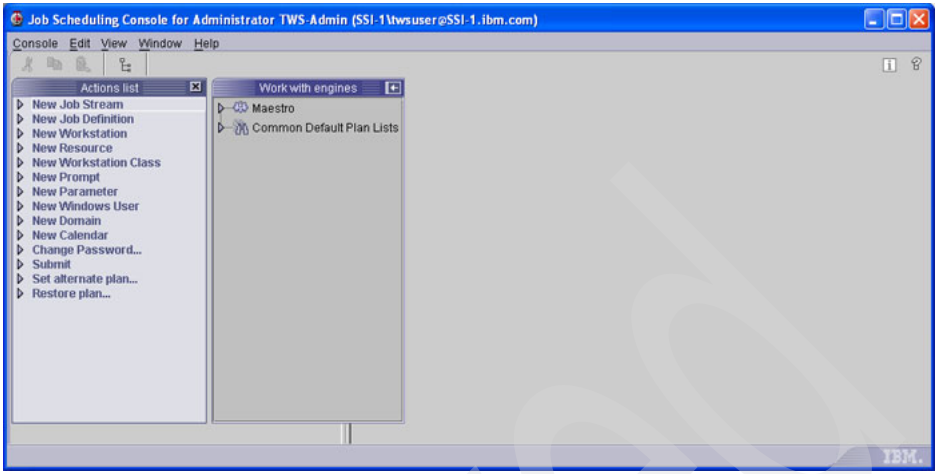

Figure 3-6 JSC main window

The Action List pane enables you to create new objects such as workstations, workstation classes, domains, calendars, jobs, and users. Figure 3-7 shows an open New Workstation list in the Action List pane, which is expanded to show all available scheduling engines. In this example, only one scheduling engine is shown with the default instance name of Maestro.

To create a new object, open the associated list entry and click on the scheduling engine name associated with the destination TWS database.

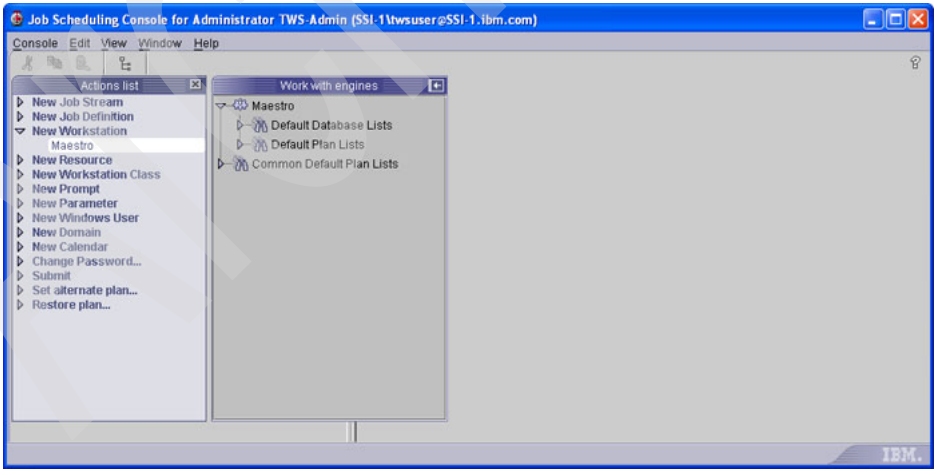

Figure 3-7 Action List pane

The JSC can access the databases only from the master or wherever the mozart directory is accessible via a TWS Connector or the composer CLI. We do not recommend mounting or mapping the mozart directory, as the TWS database should not be accessible to all users. The JSC through the TWS Connector reads the mozart directory and the corresponding file pertaining to your choice. At this point you can add, modify, copy, and delete data. Make sure you have the proper security access when performing such operations.

Open the **Default Database Lists** group to see a list for each TWS object type (Figure 3-8). Double-click any of the lists, and the JSC opens an Object List View pane for all objects associated with that list type. You can also create your own groups and filtered lists, which can be used to control the objects presented in your Object List View pane.

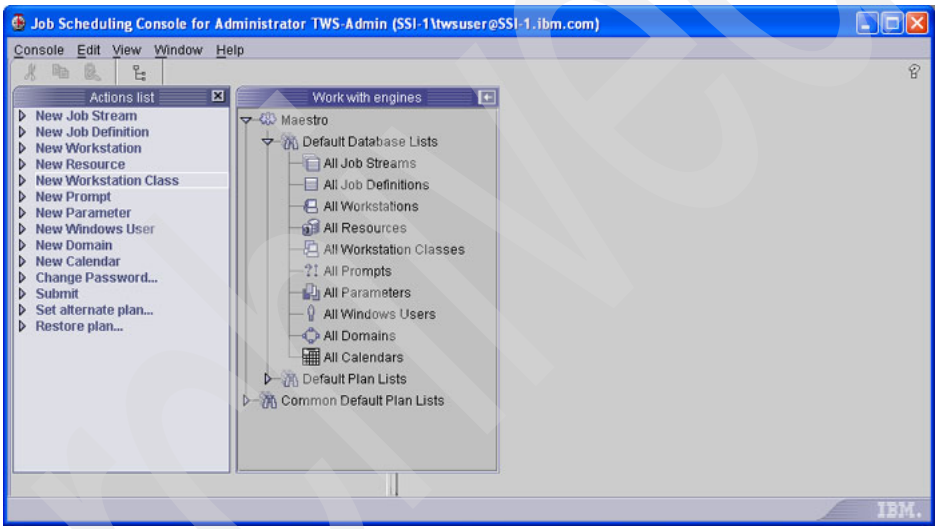

Figure 3-8 Database lists

Double-click an Object List View under Default Database Lists to display the Object List View for the associated object type. You can click the  in the upper-right corner of the Object List View or select **Console** → **Detach Task** from the JSC menu bar to detach the pane as a separate window. Figure 3-9 on page 37 shows a detached All Job Definitions list.

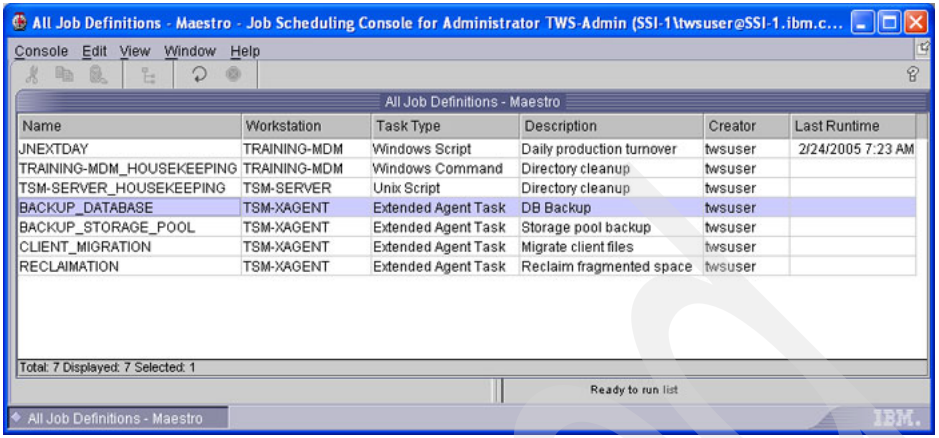

Figure 3-9 List of jobs defined on your server

You can perform modifications of job properties, deletions, and replication (Create another choice) on an existing job by right-clicking the desired job and selecting the appropriate menu item. The window in Figure 3-10 appears if you select the **Properties** menu option. For assistance with this window, click the question mark in the upper-right corner of the window and follow the instructions to make changes or review the job definition.

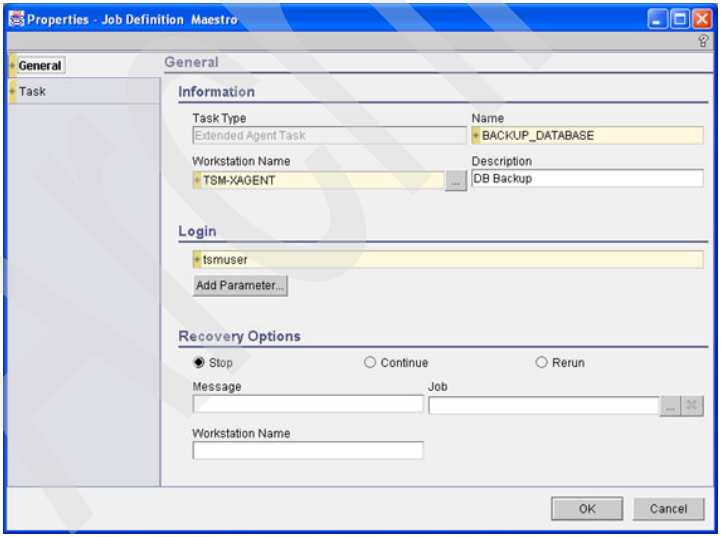

Figure 3-10 Changing job properties: General tab

The initial properties presented in the job properties window are associated with the General tab. You may also select the **Task** tab for additional job properties. Figure 3-11 is an example of the display associated with the Task tab.

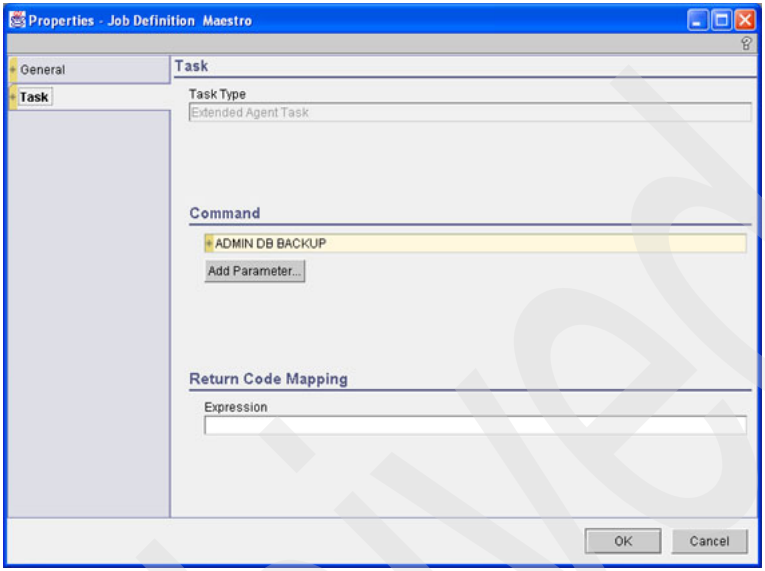

Figure 3-11 Changing job properties: Task tab

As shown, adding a new job or modifying an existing job should be relatively easy. We now offer a few explanations for you to take full advantage of TWS.

With TSM and TWS, you can take advantage of several options to ensure that your TSM jobs are successful. Recovery Options can be used to ensure that a backup is successfully executed every time. By choosing **Continue** or **Rerun**, the Recovery Job or Recovery Prompt can be used to ensure that backups are always successful. You can key in the recovery job, which for example may be another set of tapes to back up to if the primary tapes are not available, or a different directory or system if `file system full` occurs. Whichever you choose, a backup should always be successful. Also, you can use Recovery Options to send messages to the operations staff on the proper execution of a recovery job.

The Interactive option is for NT systems in your TWS environment. This option is used if the scheduled job requires manual intervention.

Opening the Default Plan Lists group presents a list for each TWS object type (Figure 3-12 on page 39). Double-click any of the lists, and the JSC opens an Object List View pane for all object instances associated with that list type. You can also create your own groups and filtered lists, which can be used to control the object instances presented in your Object List View panes.

Archived

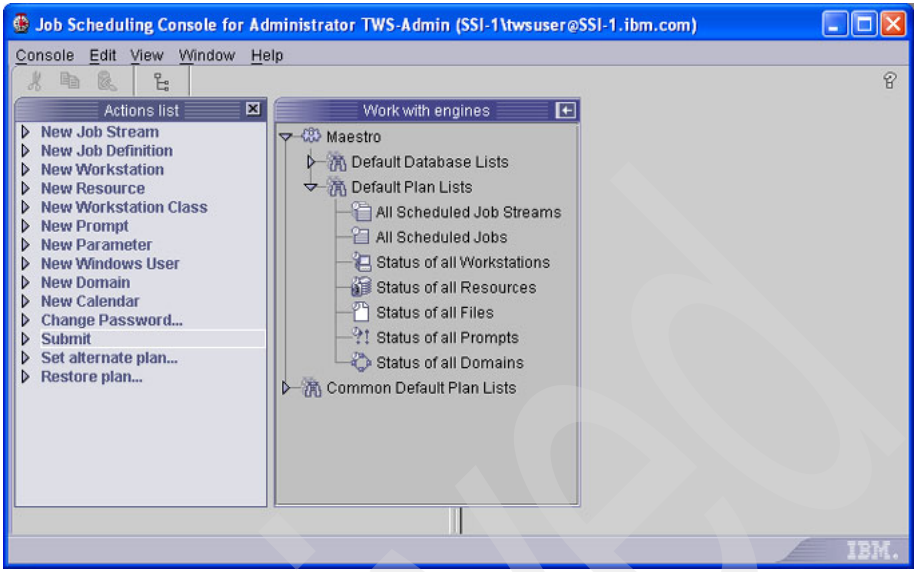

Figure 3-12 Plan lists

Double-click an object list view under Default Plan Lists to display an Object List View pane for the associated object type. Click the in the upper-right corner of the Object List View or select **Console** → **Detach Task** from the JSC menu bar to detach the pane as a separate window. Figure 3-13 shows a detached All Scheduled Jobs list.

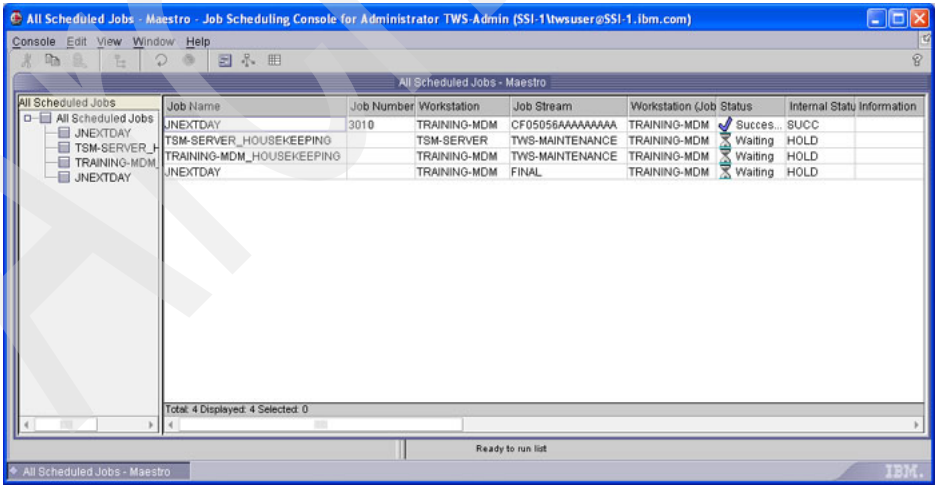

Figure 3-13 Scheduled jobs in Tivoli Workload Scheduler

The Object List View pane information about TWS scheduling objects in the current plan and well as modify some of the object attributes To access a specific object in an Object List View pane, right-click the object and choose an action.

Double-click the respective object instance list for an object type to display the corresponding information, as summarized here:

Workstations Displays status of each TWS system, including information about its connectivity with the master, the run number, the node name, and other information. The exact information can be accessed from the CLI through the **conman showcpus** command and associated qualifiers.

Domains Displays the domains, the associated domain manager, and its parent.

Job Streams Displays all job streams and their status, priority, start times, and dependencies.

Jobs Displays all jobs and their status, priority, start times, and dependencies.

Resources Displays resources and their associated workstations. Resources can be used when two or more jobs require the use of the same resource, such as a set of tapes (the tape is the resource). When one job is finished utilizing the one set of tapes or a resource, TWS releases the resource and the next job commences.

Prompts Displays the prompt's description and its associated workstation, job stream, and job.

Files Displays the files with their associated workstations, job streams, and jobs.

All of these features can be used to complement any Tivoli product, especially TSM. Each feature is discussed in more detail in Chapter 4, "Sample scenarios" on page 43.

You can see the status of the scheduled jobs by double-clicking the **All Scheduled Jobs** plan list. This shows whether the job was successful, abended, or failed. It also can show whether the job is ready and when it will start, according to start time or dependency columns, which can include resources, files, and other jobs and schedules.

If a job fails or abends, you can see the reason why it was unsuccessful by right-clicking on the job (Figure 3-14 on page 41), which opens a menu of options for viewing information about the job or taking action on the job.

Archived

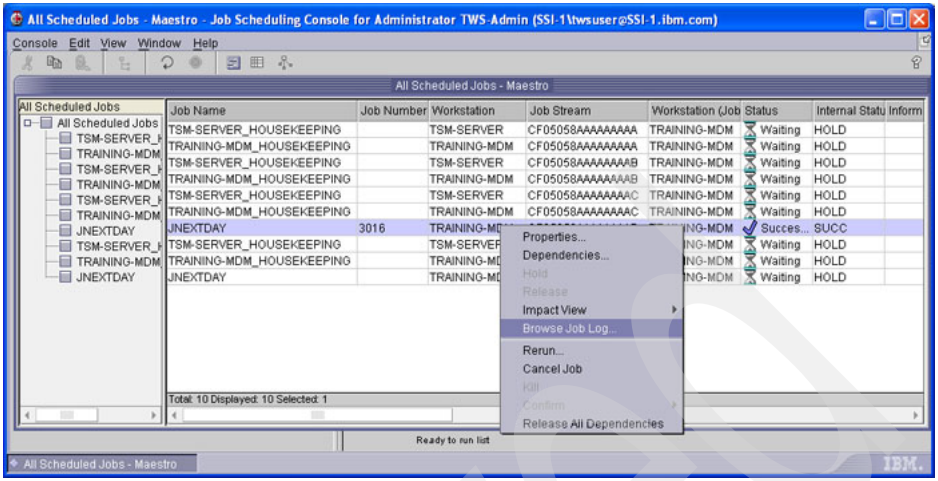

Figure 3-14 Right-click to see the status of the job

Click **Browse Job Log** in the context menu to read the log associated with that job. (located in the twshome/stdlist/MM.DD.YYYY directory). For more about various other options, refer to the *IBM Tivoli Workload Scheduler 8.2 Planning and Installation Guide,* SC32-1273. Figure 3-15 shows a sample stdlist output.

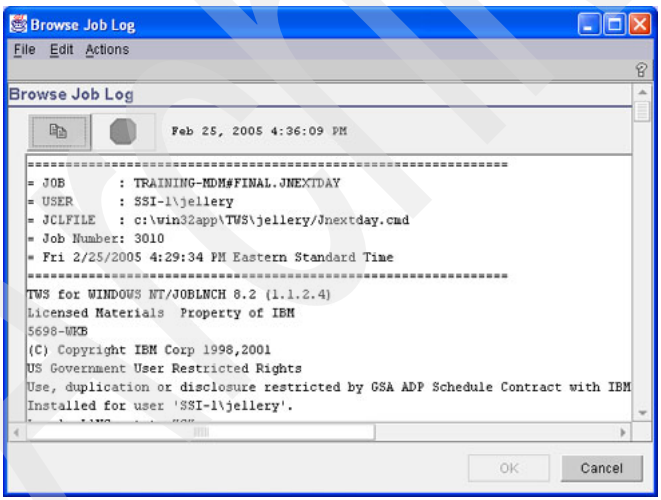

Figure 3-15 Status of the job

The stdlist header contains the job name, its associated schedule and CPU names, as well as the logon name and the script location. As shown, it also displays the process ID, and the date and time of the job execution.

The body of the stdlist contains the steps executed for the job, whether successful or unsuccessful. The information contained depends solely on its script and the information to standard out (stdout). You can print this output.

3.3 Summary

In this chapter we described the following TSM functions that can be scheduled with the Tivoli Workload Scheduler:

- Back up database: To back up a TSM database to sequential access volumes.
- Back up storage pool: To back up primary storage pool files to a copy storage pool.
- Reclamation: To reclaim a volume, based on the percentage of reclaimable space on a volume.
- Migration: To copy files to a library when a threshold is reached on the disk.
- Back up device configuration: To back up the device class, library, and drive definitions.
- Back up volume history: To back up sequential volume history information to one or more files.
- Expire inventory: To manually start inventory expiration processing.
- Restore database: To restore a database or volume to its most current state.

The information in this chapter is especially useful if you are not familiar with TSM scheduling operations.

Archived

4

Sample scenarios

This chapter address several scenarios when utilizing Tivoli Workload Scheduler with the Tivoli Storage Manager product sets.

We demonstrate how to schedule TSM functions in TWS. Primarily, we create jobs and job streams in TWS to handle such functions. We also address dependency options and display jobs and job streams output.

This chapter has the following sections:

© Copyright IBM Corp. 2005. All rights reserved.

4.1 TSM Extended Agent solution components

The TSM Extended Agent solution uses operational components such as:

- Tivoli Workload Scheduler master (master domain manager)

  Contains the TWS scheduling database and manages the TWS current plan.

- Tivoli Workload Scheduler domain manager

  Enables distribution of dependency resolution and such in the TWS topology.

- Tivoli Workload Scheduler Fault-tolerant Agent

  Hosts the TSM Extended Agent (XA) and is responsible for such tasks as triggering jobs on schedule. Note that any domain manager or the master domain manager could also be used to host a TWS extended agent.

- TSM Extended Agent

  Provides an enhanced TWS job command interface for executing TSM commands on a TSM server.

- Tivoli Storage Manager server

  Executes TSM administration commands and schedules execution of commands on TSM clients.

- Tivoli Storage Manager client

  Executes TSM commands and non-TSM batch files as directed by the TSM scheduler on the TSM server.

Figure 4-1 shows the relationships between these components.

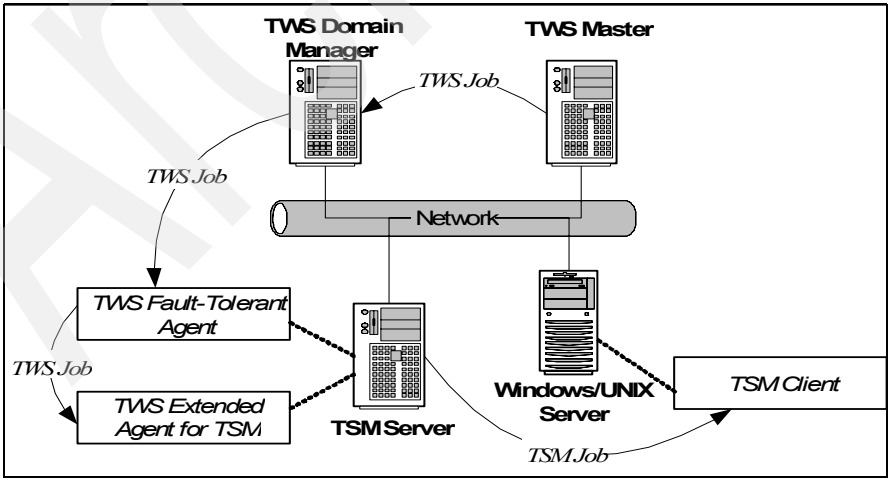

Figure 4-1 TSM Extended Agent components

Archived

4.2 Tivoli Storage Manager command execution

This section walks you through the process of a TSM job executing through the TSM Extended Agent. Figure 4-2 on page 46 provides a view of the components involved with job execution and how they interact with each other.

1. The TWS Fault-tolerant Agent hosting the TSM Extended Agent determines that all dependencies have been met for the TWS extended agent job, so the job is released for execution (the extended agent method script is called).
2. The TWS extended agent method script executes, parsing the command to be executed by TSM and calling TSM to execute the command.
3. ADMIN commands are executed directly by the TSM server that the TSM Extended Agent is installed on.
4. CLIENT commands are scheduled via the TSM scheduler, either directly by the method script via the **`Define Schedule`** or **`Define Association`** commands, or indirectly by the TSM **`Define ClientAction`** command (this causes TSM to schedule the execution of the CLIENT command with the TSM scheduler). After the command has been scheduled with or by TSM scheduler, the following actions are performed:
   a. The TSM scheduler contacts the TSM client Acceptor service on the TSM client to start the Task Scheduler service.
   b. When the Task Scheduler service is up and running, the execution of the scheduled command starts. Note that the TSM scheduler reports that the schedule is in pending state until execution of the command is started.
   c. When command execution has completed, the TSM client passes back the result (return code) from the command (TSM client action or Windows batch file).
   d. If the Client command was executed via the TSM **`Define ClientAction`** command, the result is returned to the XA script that has been waiting. Otherwise the XA has been polling the TSM server for event status and will exit the polling loop when TSM returns a status of Completed or Failed, or the polling count reaches the time-out limit. By default, the polling count is 0, indicating “poll forever” (no time-out).
5. The result of the job is stored on the extended agent host workstation, and the job log is available via the Job Scheduling Console (JSC).

Archive

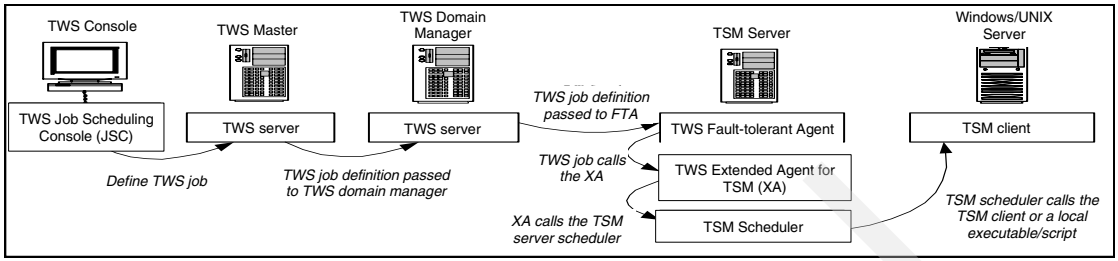

Figure 4-2 Tivoli Storage Manager command execution

4.3 When to use the TSM Extended Agent

Use the TSM Extended Agent to schedule the execution of TSM commands on a TSM server or TSM client. The TSM Extended Agent is particularly useful for:

- Scheduling batch files to run before or after TSM commands. (Use TWS job dependencies.)
- Scheduling TSM commands that have to be run in a specific order. (Use TWS job dependencies to specify the job order).
- Limiting the number of TSM backups, and so on, that TWS can run at a time. (Use TWS special resources to limit this.)
- Scheduling TSM backups on a new TSM client.
- Scheduling backups on multiple TSM clients using a single job definition.

4.4 TSM Extended Agent installation

1. Install a TWS Fault-tolerant Agent on the TSM server. Refer to the *IBM Tivoli Workload Scheduler 8.2 Planning and Installation Guide,* SC32-1273 for detailed instructions for installing a Fault-tolerant Agent.
2. Copy the TSM Extended Agent method script tsmxagent and the supporting script PARMS into the methods directory of the TWS Fault-tolerant Agent.
3. On the TWS Fault-tolerant Agent, create a UNIX user ID (referred to as *uid*) to execute TWS jobs using the TSM Extended Agent. This user ID must be assigned the primary group of `tivoli` and the password must be set.
4. On the TWS Fault-tolerant Agent, execute:

   ```
   chown <tws_uid>:tivoli tsmxagent PARMS
   ```

   Change the owner of the TWS XA for TSM and PARMS scripts to the *tws_uid* that owns the TWS instance (`twsuser`) and the group `tivoli`.

5. On the TWS Fault-tolerant Agent, secure the scripts by executing:

   ```
   chmod 750 script_name PARMS
   ```

6. On the TWS Fault-tolerant Agent, update the TWS Security file to authorize *uid* to use the TWS **parms** utility:

   a. Log on as *tws_uid*.

   b. Run **dumpsec > Security.conf**

   c. Edit Security.conf to add *uid* to the CPU=@+LOGON=*tws_uid*,root,... statement

   d. Run **makesec Security.conf**

7. On the TWS Fault-tolerant Agent, set the TSM admin user ID that will be used by the extended agent. Setting the admin user ID can be accomplished in one of two ways. If you need to change the default TSM admin user ID used by the tsmxagent, change the USERID=*tsm_uid* statement so that *tsm_uid* is the TSM user ID you want to be the extended agent default. An option for overriding the default tsmxagent USERID is to create a tsmxagent.opts file in the home directory of the UNIX user ID you specify in your TWS job definition. This file must contain an entry that looks like:

   ```
   tsmAdmin tsm_uid
   ```

   tsm_uid is the TSM user ID you want the TSM Extended Agent to use to execute TSM commands. Be sure to run **chmod 600 tsmxagent.opts** to secure this file.

8. Set a password for each TSM admin user ID you plan to use: Log on as *maestro_uid* and execute **methods/PARMS *tsm_uid password*** for each TSM admin user ID that will be used with the TSM Extended Agent.

9. Use the TWS Job Scheduling Console (JSC) to create a workstation instance for the TSM Extended Agent. When you have created the TWS extended agent workstation definition, Jnextday has run at least once, and the hosting TWS Fault-tolerant Agent is linked, you can schedule TSM jobs.

Archive

4.5 Creating job definitions

To create a TSM Extended Agent job definition using the JSC, click **Extended Agent Task** in the Action list pane and fill in the Name, Workstation, and Login fields. Click the **Task** tab and fill in the Command field (Figure 4-3).

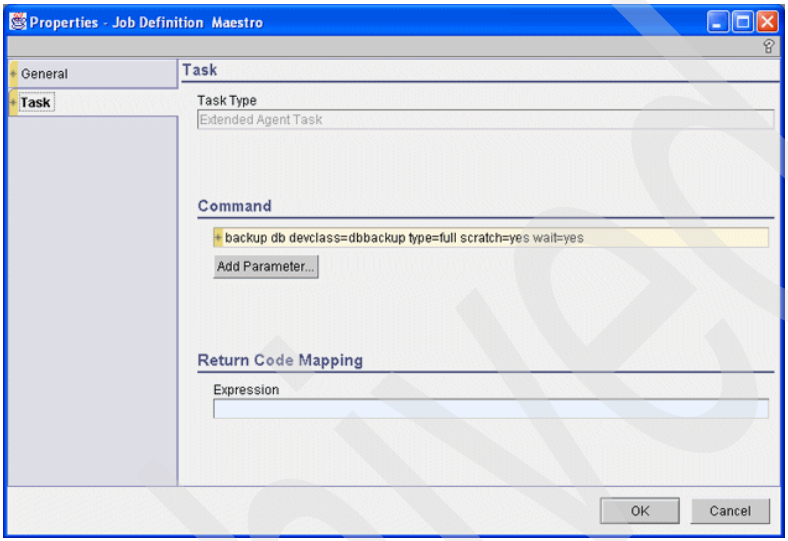

Figure 4-3 Job Properties Task tab

4.5.1 Tivoli Storage Manager client command strings

Example 4-1 shows the syntax that is used to define TSM client jobs. Using the CLIENT keyword and the listed options, you can build TSM Extended Agent jobs that will execute on a TSM client.

Example 4-1 Tivoli Storage Manager client command string

```
CLIENT -a <action> -d <policy domain> -n <TSM node name> [-p value] [-s <value>
[-t <value> ] [-- <TSM options>]

-a <action>        Tivoli Storage Manager Define Schedule action (see Tivoli
                   Storage Manager Administrators Reference)
-d <policy domain> Tivoli Storage Manager Policy Domain
-p <polling interval>Time to wait before the next status poll
-n <TSM node name> Name of the Tivoli Storage Manager client to execute the
script on
-s <script path\\name>The script to exec when the "-a command" is specified.
Use '\\'
                   instead of '\'. Due to script name parsing, avoid using "
-" in the
```

Archived

```
                    script path\name.
-t <timeout count>  Number of polling intervals before timing out. Defaults to
0 = Poll
                    Forever
-w                  Execute the client command synchronously Commands will not
                    time-out so the "-p" & "-t" options are ignored.
-- <TSM options>         Any Tivoli Storage Manager options you need to add to
the Tivoli
                    Storage Manager Define Schedule command to be executed. If
                    you use the "--" option it MUST be the LAST option in the
job
                    command string. When multiple values are required for a
Tivoli
                    Storage Manager option, enclose the values in escaped
double
                    quotes. e.g. \"<values>\"
```

Example 4-2 Command string to execute a windows BAT file asynchronously

```
CLIENT -a command -d CLXP_TEST_DM -p 30 -s C:\\Program
Files\\Tivoli\\TSM\\baclient\\SCRIPTS\\tsm_rc_0.bat -n CBUEHLER1
```

Example 4-3 Command string to execute a UNIX script asynchronously

```
CLIENT -a command -d CLXP_TEST_DM -w -s /home/maestro/script_name -n CBUEHLER1
```

Example 4-4 Command string for incremental backup with options

```
CLIENT -a incremental -d CLXP_TEST_DM -p 600 -n CBUEHLER1 --
OPTIONS=\"-nojournal -optfile=dsm_SQLLOG.opt\"
```

4.5.2 Tivoli Storage Manager Admin command strings

To execute administration commands on a TSM server, use the following syntax to generate the command strings for TSM Extended Agent jobs:

ADMIN *Tivoli Storage Manager Admin command with options*

When entering a double quote (") in a TSM **Admin** command, be sure to precede it with a backslash (\") because the JSC uses double quotes as input delimiters. Some TSM administration commands require the WAIT=YES option; otherwise they will schedule an action and immediately return an exit status of 0. This will result in the associated TWS job going to the success state prior to the execution of a TSM administration command.

Example 4-5 Command string to create a backup

```
ADMIN generate backupset cbuehler2 bkpset dev=clxp_test_tape ret=10 wait=yes
```

Important: When scheduling TSM commands that start before midnight and end after midnight, use the -w *client* or wait=yes option to avoid status polling problems related to the TSM scheduler cleaning up old schedules at midnight.

4.5.3 Scheduling jobs using the workstation class

Creating multiple job definitions and a job stream to back up each server that uses the TSM Extended Agent for backup can lead to major maintenance overhead in an environment of any significant size. To address this, the TSM Extended Agent has been updated to override the -n TSM_NODE_NAME parameter based on the workstation name. This enables schedules to be created using the TWS workstation class feature that executes a job stream against each workstation in the workstation class definition. This means that a single job stream and set of job definitions can support any number of systems that have identical backup requirements. To use this new feature of the TSM Extended Agent, follow these instructions:

1. Define extended agent workstation definitions.

   Define an extended agent workstation for each system you will be running backups on. The workstation name must begin with XA_ and end with _<TSM_NODE_NAME>. For example, an XA workstation definition for CBUEHLER9 could be XA_CBUEHLER9 or XA_WIN_CBUEHLER9.

   Note: You can automate this with scripting and the TWS composer interface.

2. Create a TWS workstation class.

   Create a TWS workstation class that includes each system you will be running a common set of backup jobs against using a single job stream. This can be done via the TWS Job Scheduling Console.

   Note: You can automate this with scripting and the TWS composer interface.

3. Create the TSM Extended Agent job definitions.

   Define the jobs you will use to back up the systems (Figure 4-4 on page 51), specifying `Workstation Class Command` as the Task Type and the workstation class created in “Define extended agent workstation definitions.” on page 50.

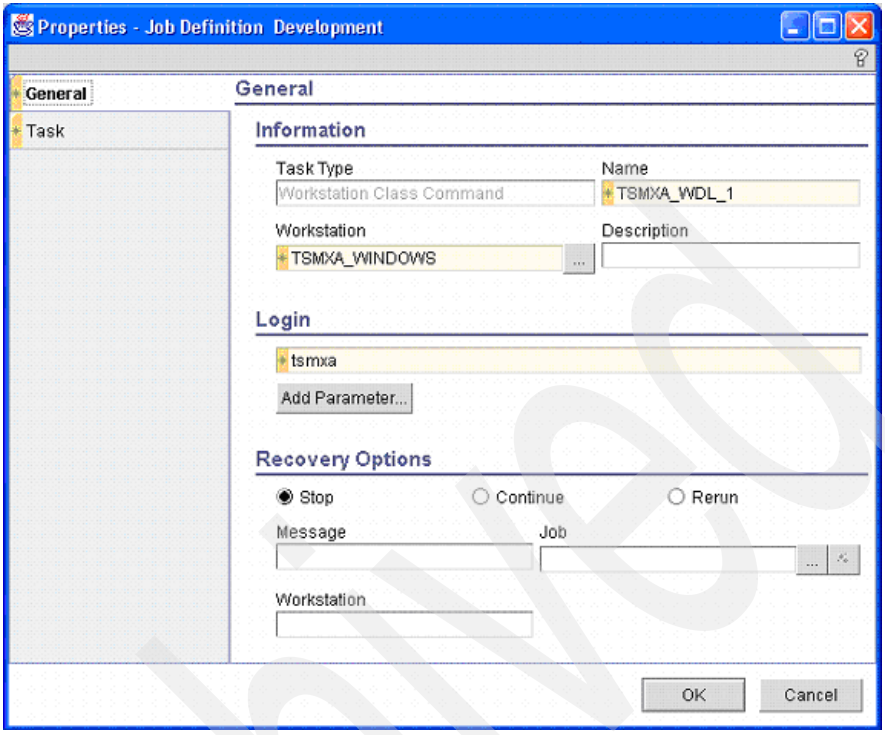

Figure 4-4 Create the TSM Extended Agent job definitions

Note that the -n BOGUS01 parameter specified in the Command field of the Task tab will be overridden by the <TSM_NODE_NAME> part of the XA workstation name, as shown in Figure 4-5 and Figure 4-6 on page 52.

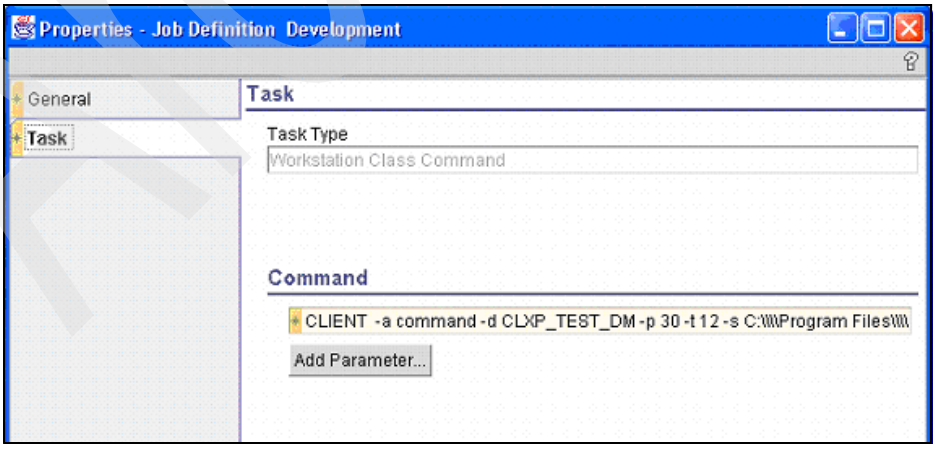

Figure 4-5 Task definition: part one

Archived

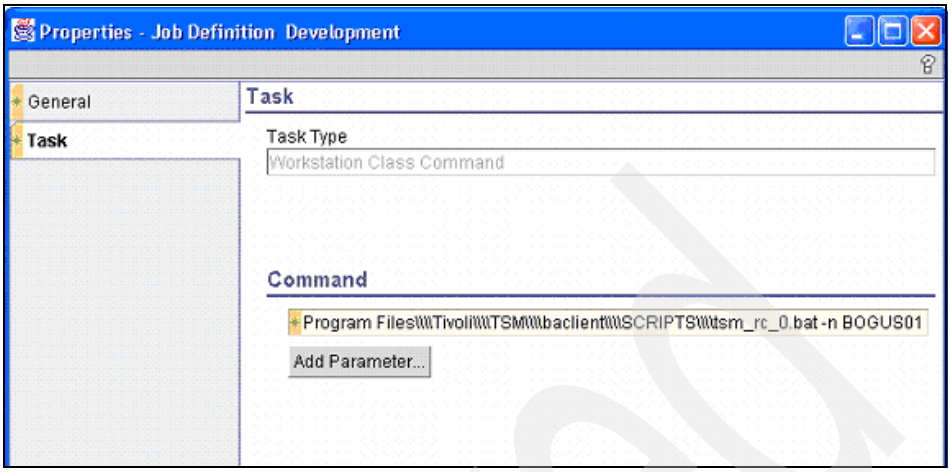

Figure 4-6 Task definition (continued from Fig 4-5)

4. Create the job stream.

   Create a job stream using the workstation class name in the Workstation field as in Figure 4-7.

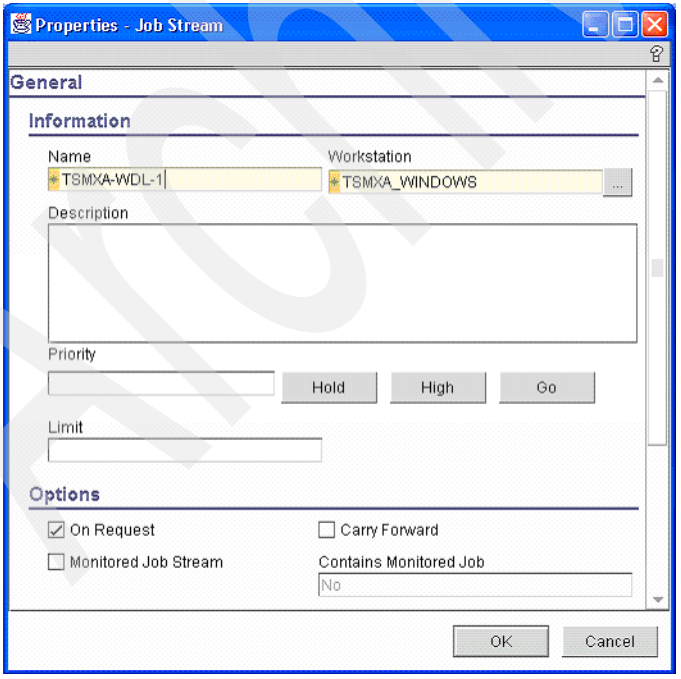

Figure 4-7 Create the job stream

Archived

5. Add and link your Workstation Class Command jobs as in Figure 4-8.

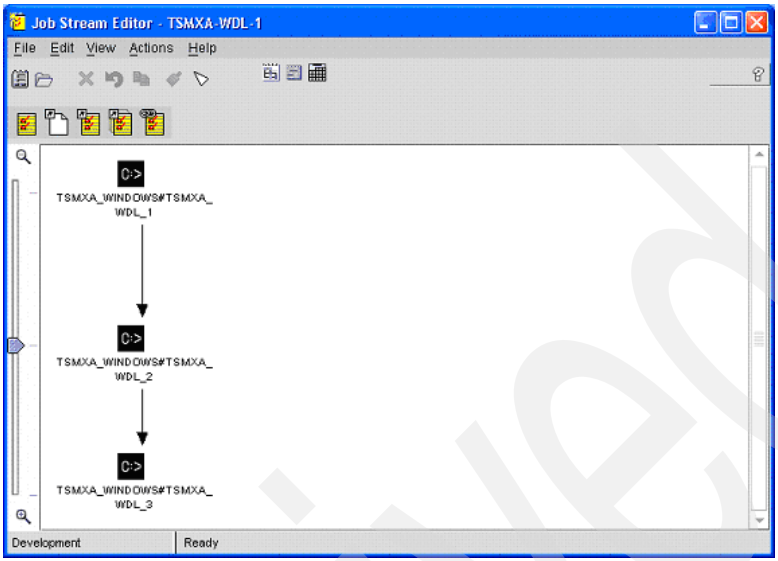

Figure 4-8 Add and link your jobs

Monitoring workstation class job streams

When you submit a workstation class job stream, it generates a job stream instance for each workstation defined as part of the workstation class.

For example, if you submit a workstation class job stream with an alias of J001, an instance of the job stream will be created for the XA_W1_CBUEHLER9 workstation and another for workstation XA_W2_CBUEHLER9.

Monitoring workstation class jobs

Each workstation class job stream instance is comprised of a set of job instances. The *dependency chains* for these jobs are resolved by workstation. For example, this means that the TSMXA_WDL_1 job running on XA_W1_CBUEHLER9 must complete before the TSMXA_WDL_2 job will run on XA_W1_CBUEHLER9.

Furthermore, the execution of the TSMXA_WDL_2 job on XA_W2_CBUEHLER9 does not depend on the execution of TSMXA_WDL_1 on XA_W1_CBUEHLER9. In the event that a job in a dependency chain fails, it can be rerun or manually set to SUCC state without affecting the execution of the backup job stream on other workstation-class workstations.

Archived

4.6 Description of the scenarios

These scenarios involve information about backup databases, backup storage pools, migration, and various other typical scenarios.

4.6.1 Database backup

Our first scenario is a database backup. You can specify the type of backup that your company requires. This depends on the necessary data's level of availability for recovery purposes, as well as addressing service level agreements (SLAs) between your company and third-party vendors.

We start with scheduling a TSM function in TWS:

1. Execute the Tivoli Workload Scheduler JSC and expand **New Job Definition** in the Action list (Figure 4-9).

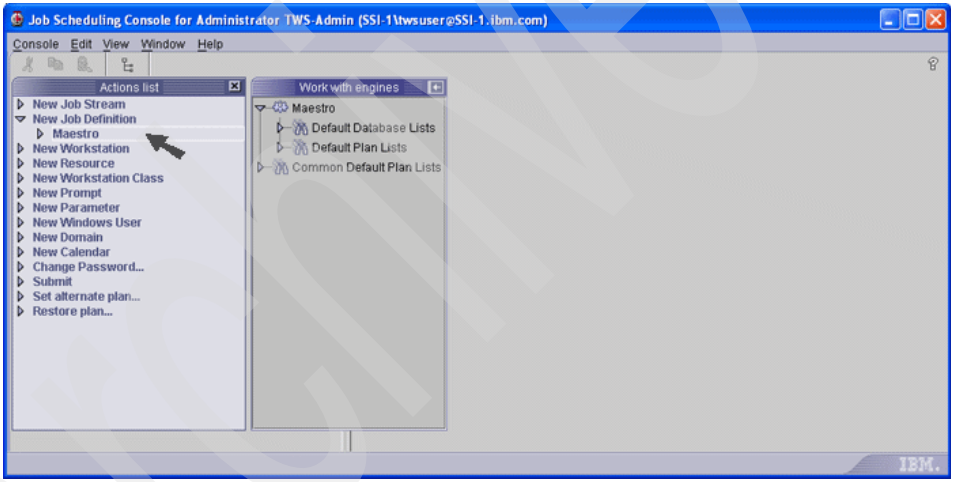

Figure 4-9 JSC Action list: New Job Definition

Archived

2. Expand the **Maestro** scheduling engine and click **Extended Agent Task** (Figure 4-10).

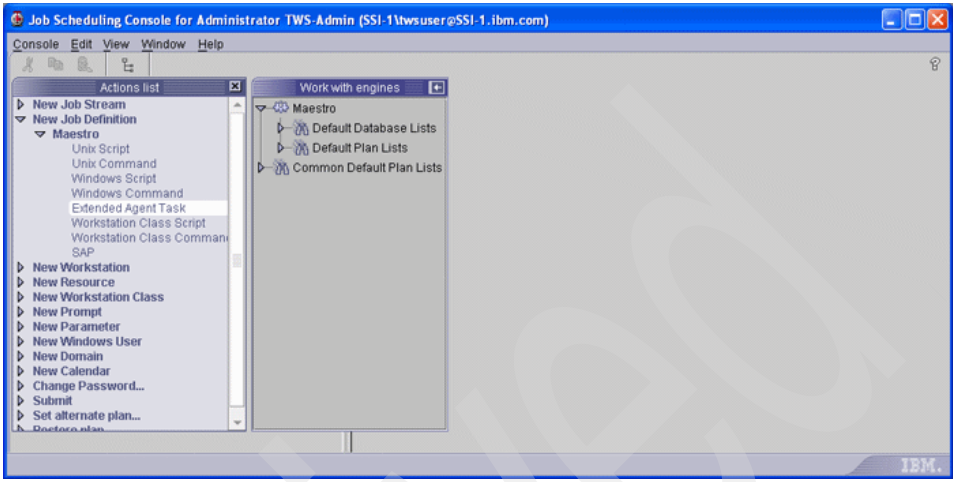

Figure 4-10 JSC: new Extended Agent Task

3. This opens the job's Properties window (Figure 4-11).

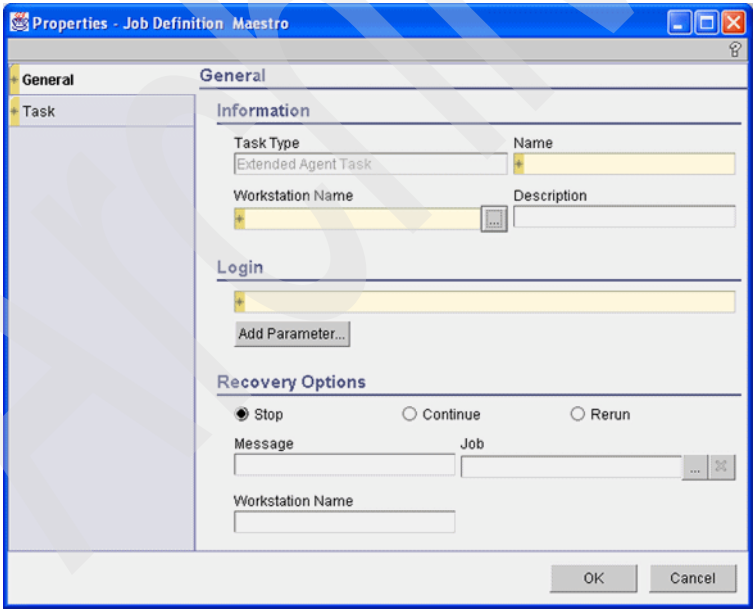

Figure 4-11 New job properties: General tab

4. Click the ellipsis (**...**) next to the workstation field to open the Find Workstation window (Figure 4-12). Select the workstation definition in which the new job will reside. In TWS, a workstation must be identified with the job or where the backup script resides.

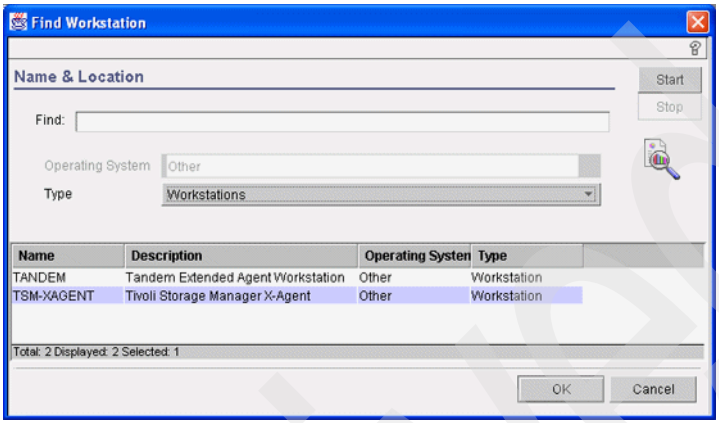

Figure 4-12 Select the workstation

5. Complete the necessary fields in the Properties window (Figure 4-13). `Login` refers the user ID that will execute the job (script or command). `Recovery Options` is an important field to consider. When a job does not complete successfully (or the status is ABEND), the entered Recovery Job is executed.

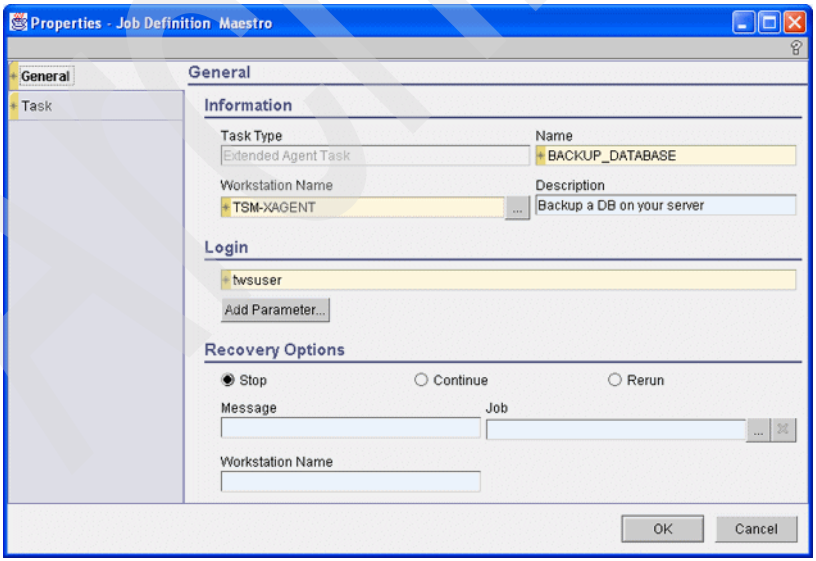

Figure 4-13 Job properties window for database backup: General tab

6. Designate the script name or the command, ensuring that the path is included or the .jobmanrc is present in the user's directory (Figure 4-14). The path can also be a parameter if necessary. See *IBM Tivoli Workload Scheduler 8.2 Planning and Installation Guide*, GC32-1273, and *IBM Tivoli Workload Scheduler Job Scheduling Console 1.3 User's Guide*, GC32-1257 for more about the Parameter object.

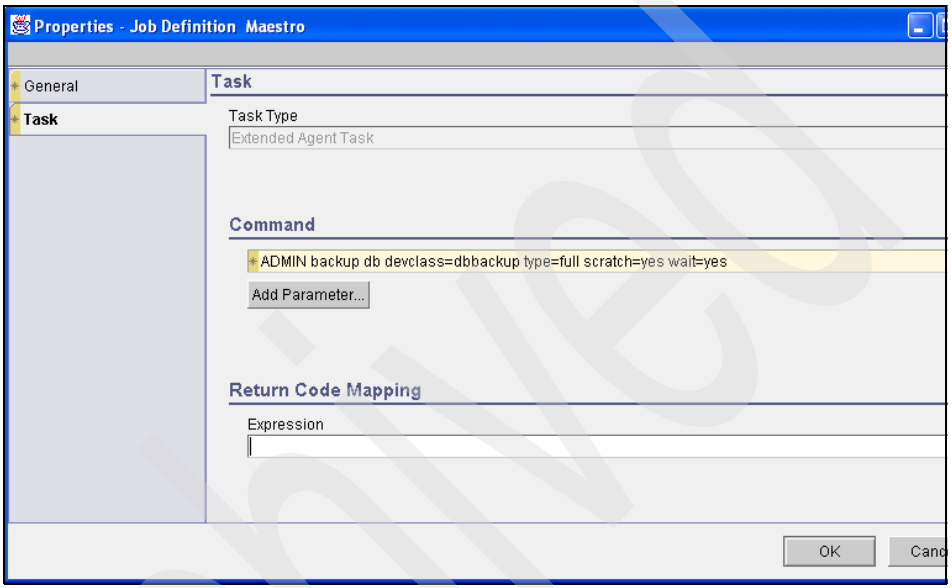

Figure 4-14 Job properties window for database backup: Task tab

7. When you are finished specifying job properties, click **OK** to close the window.

Archived

8. As jobs are entered in the database, utilizing the JSC Groups and Lists feature (Figure 4-15) can help you search for a specific job.

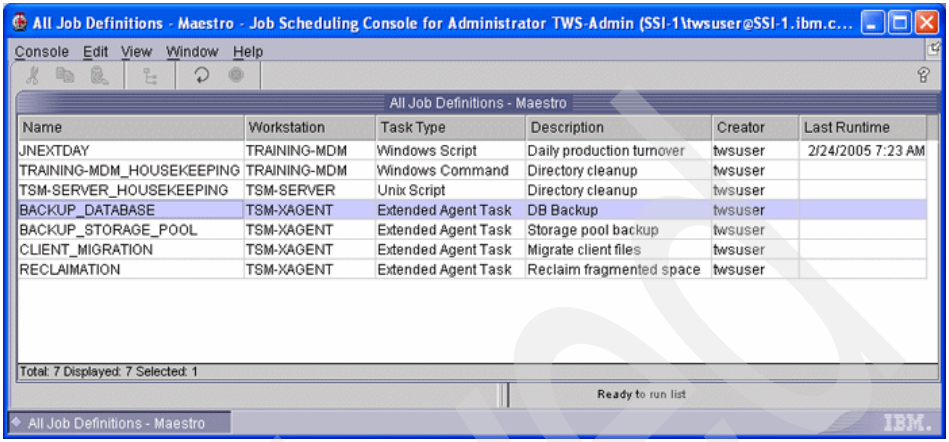

Figure 4-15 List of jobs, including BACKUP_DATABASE

When a job is defined, it must be entered into an existing or new job stream. The following steps assist you with job stream definitions:

1. For a new job stream, expand **New Job Stream** in the Action list and click the scheduling engine name associated with the destination TWS database (Figure 4-16).

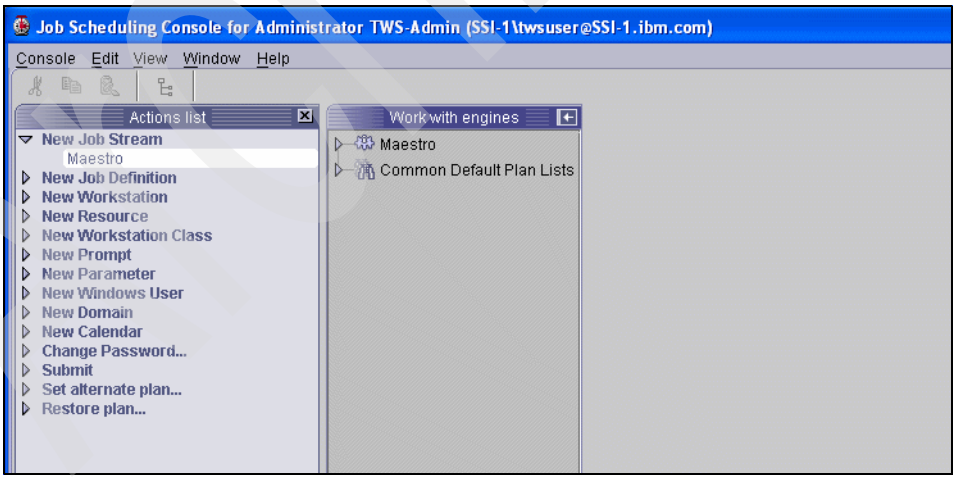

Figure 4-16 Selecting schedules to add a new schedule

2. Creating a new job stream first presents a job stream Properties window (Figure 4-17). Specify the name of the job stream and the hosting workstation name. If you are not sure of the appropriate workstation, click **...** to the right of the Workstation Name field. Naming convention methodology is highly recommended for easy monitoring and ad hoc scheduling.You can also specify the job stream's priority, limit, options, and free days.

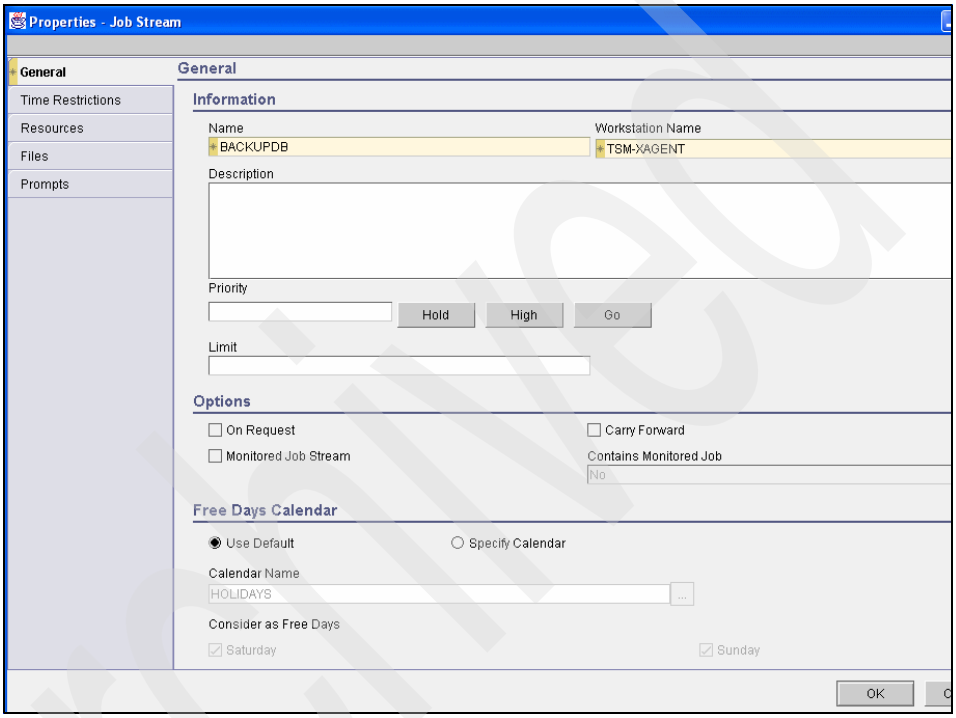

Figure 4-17 Job Stream Properties for new BACKUPDB stream

Archive

3. On the Time Restrictions tab (Figure 4-18), enter the schedule's start time, latest acceptable start time, and termination deadline time, and any required delays. Note that Days offset cannot be used at the job and job stream level. If your site has the Time Zone feature enabled, you can also specify a time zone that applies to all three times; otherwise the time zone of the job stream workstation is used.

   You also can select the action you want TWS to take if the job stream does not start by the specified time. The default action is *Suppress* the execution of the job stream and any jobs in the stream. You can also select the Continue option, which takes no action but makes entries in the Tivoli Workload Scheduler log and event files. The third option is to cancel the stream, which releases any downstream jobs and job streams from dependency on the BACKUPDB job stream or any jobs in the stream.

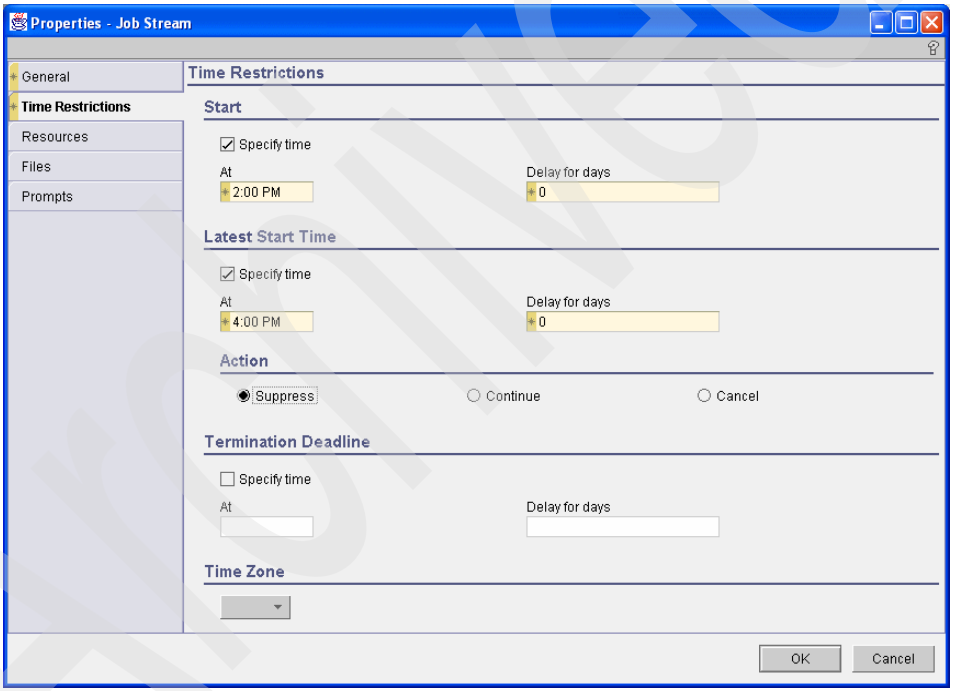

Figure 4-18 BACKUPDB Job Stream Time Restrictions

Archive

4. Click the **Resources** tab (Figure 4-19). Resources represent physical or logical scheduling resources that can be used as dependencies for jobs and job streams. The resource may be an available device (a tape or tape device). If other job streams or jobs are dependent on the same device, you may use the resource option to run jobs or job streams in a particular sequence.

   To add a resource dependency, click the green plus (**+**) sign in the upper-right corner of the Resources pane. This adds a new row to the table, where you can select the resource name, the resource workstation, and the quantity required by the job or job stream.

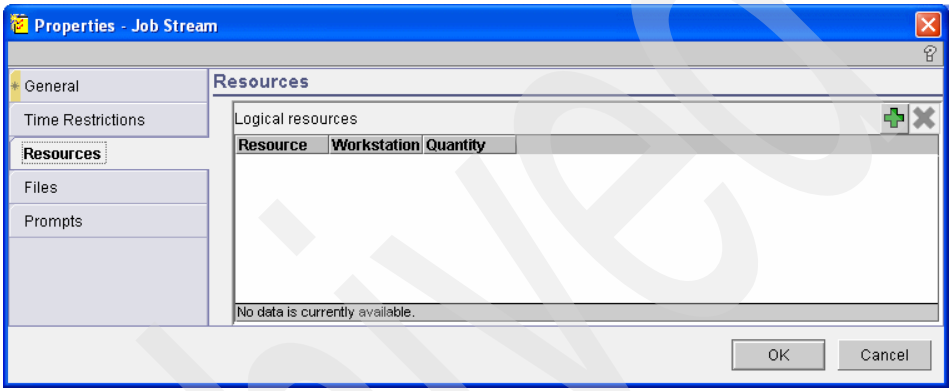

Figure 4-19 Adding job stream resource dependencies

5. On the Files tab (Figure 4-20), you do not have to enter a file as a dependency for our scenario, so leave this blank. However, adding a file dependency works the same way as adding a resource dependency.

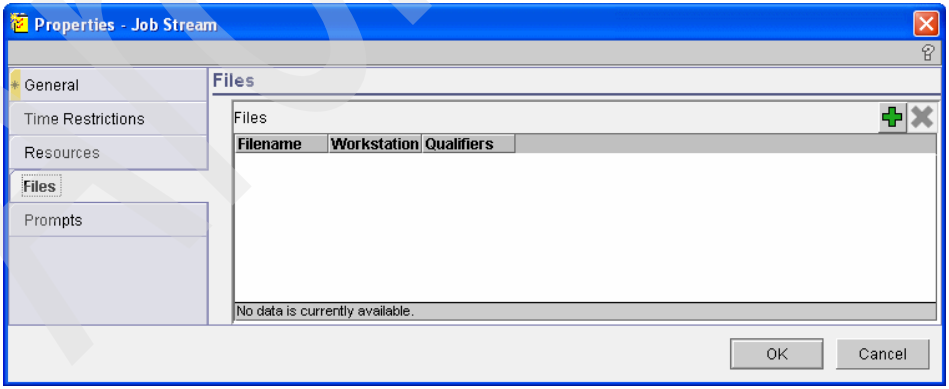

Figure 4-20 Adding job stream file dependencies

Archived

6. On the Prompts tab (Figure 4-21), you do not have to enter a prompt as a dependency for our scenario, so leave this blank. However, adding a prompt dependency works the same way as adding a file or resource dependency.

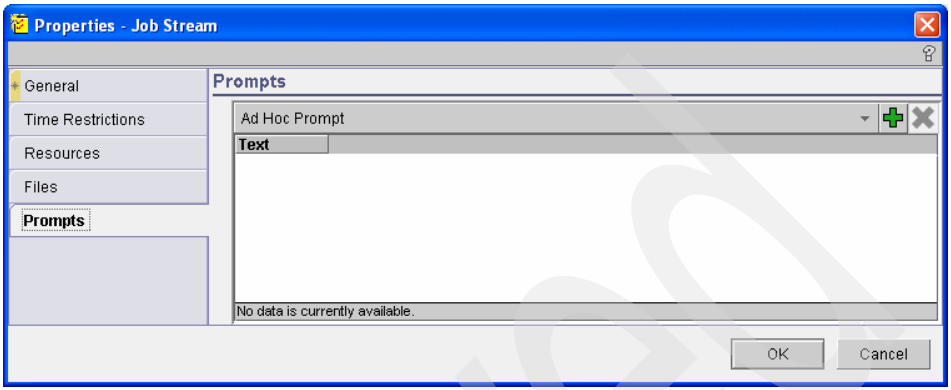

Figure 4-21 Adding job stream prompt dependencies

7. After the correct information is entered for the job stream properties, click **OK** to display the graphical view of the Job Stream Editor. In this view, you can add jobs to the job stream, define dependencies between the jobs, and define dependencies between jobs in the current stream and external jobs and job streams.

   To add a job, click the add icon indicated by the arrow in Figure 4-22.

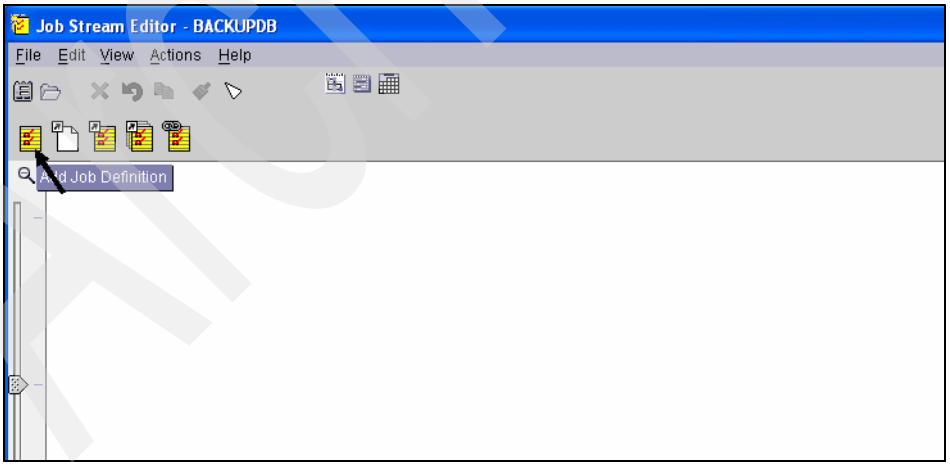

Figure 4-22 Job Stream Editor view for BACKUPDB

8. This opens a job properties window (Figure 4-23) that resembles the job streams Properties window, with General, Time Restrictions, Resources, Files, and Prompts tabs for specifying the properties of jobs in this stream. In this scenario, we simply specify the job's name and workstation name.

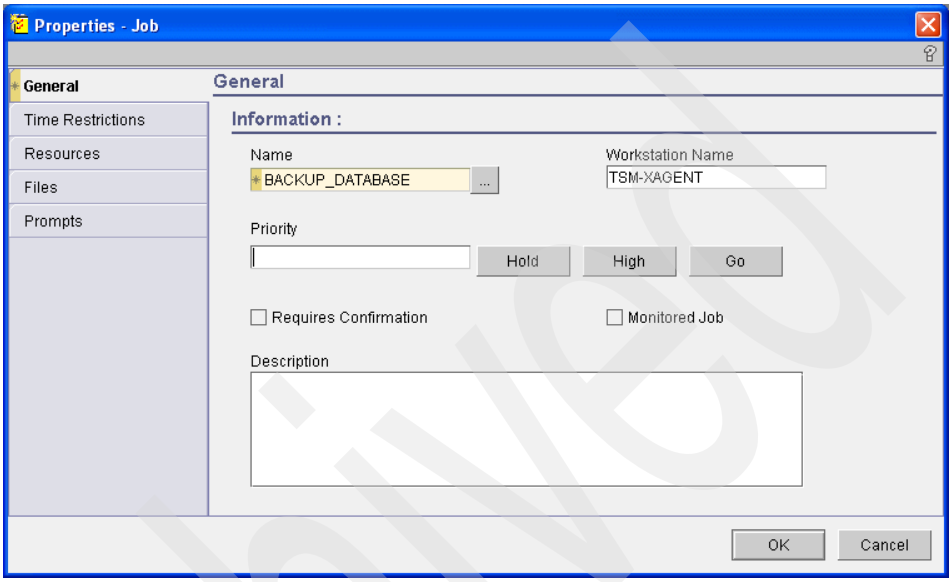

Figure 4-23 Job Properties window for BACKUP_DATABASE

9. Click **OK** to complete the process of adding the job to the job stream. The job is represented by an icon in the graphical view of the Job Stream Editor (Figure 4-24).

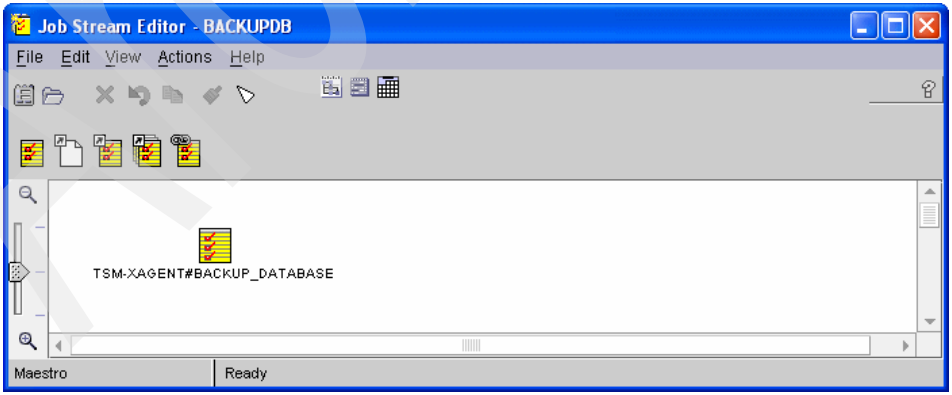

Figure 4-24 Job Stream Editor graphical view of the BACKUPDB stream

Archived

10. Selecting the days of the week, month, or year that the job stream will execute requires the definition of a run cycle. Run cycles are made up of one or more rules that either include or exclude dates. The rules are defined using the Run Cycle view of the Job Stream Editor. To select the run cycle view, click the Run Cycle icon identified by the arrow in Figure 4-25.

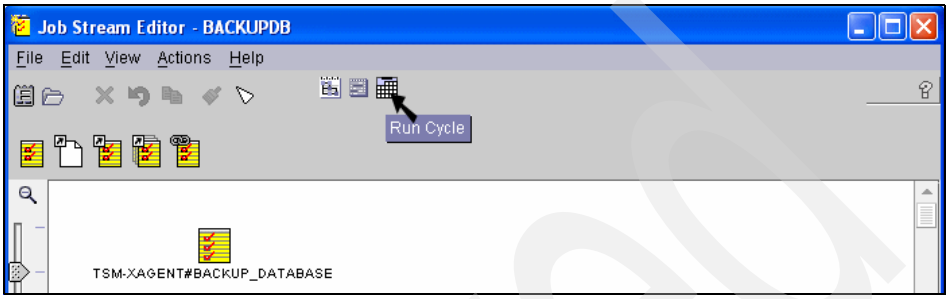

Figure 4-25 Selecting the Run Cycle view

For this scenario, we add one inclusive rule to the run cycle definition for the BACKUPDB stream. The rule selects the stream for execution every day of the year. To add the rule, click the Weekly Run Cycle icon in the run cycle view, as shown in Figure 4-26.

Figure 4-26 Adding a Weekly Run Cycle rule

11. We specify an Inclusive rule, so that the days specified by the rule cause TWS to select the stream (Figure 4-27).

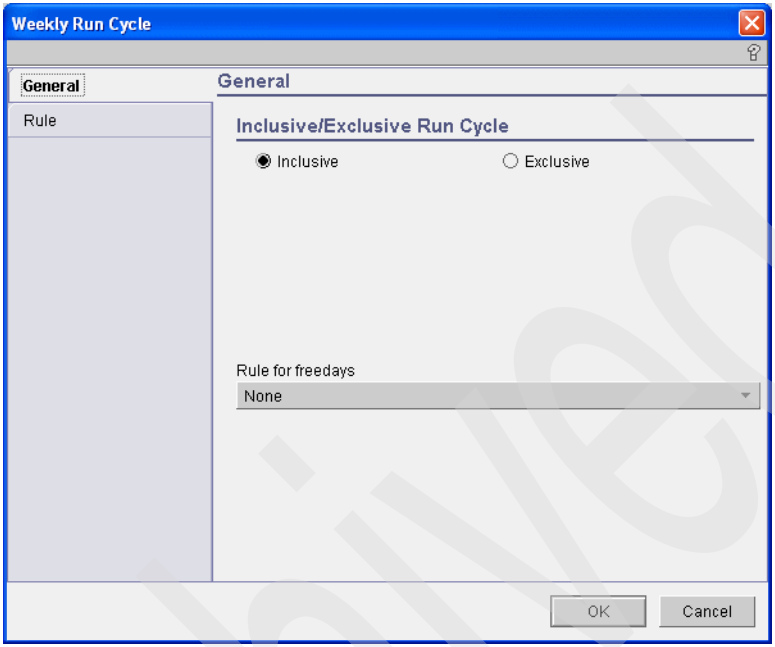

Figure 4-27 Specifying an Inclusive Run Cycle rule

Archived

12. Select the **Everyday** rule and click **OK** (Figure 4-28). This action causes TWS to execute the job stream seven days a week.

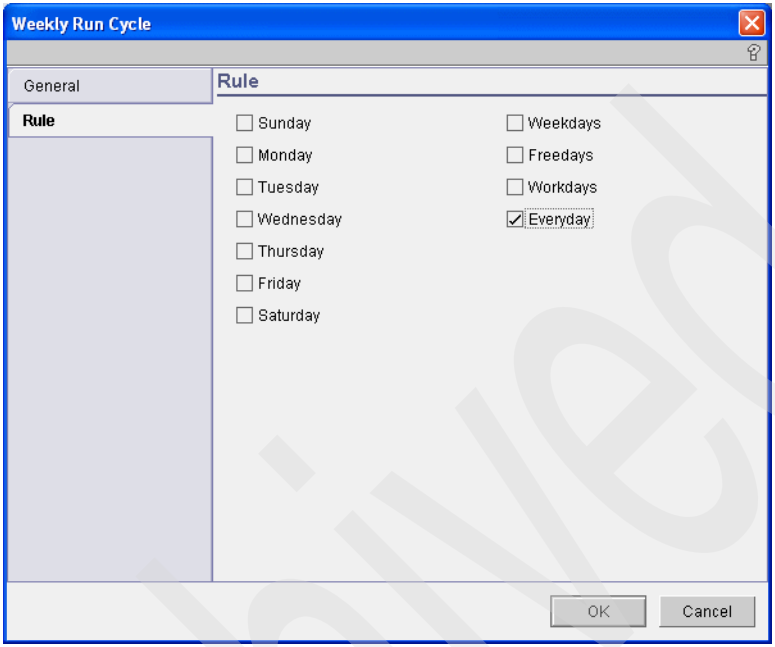

Figure 4-28 Specifying an Everyday run cycle rule

Archived

13. Figure 4-29 shows that the dates selected by the combination of all inclusive and exclusive rules are highlighted in light blue. Select **File** → **Save**, then **File** → **Close** to save the job stream and close the Job Stream Editor.

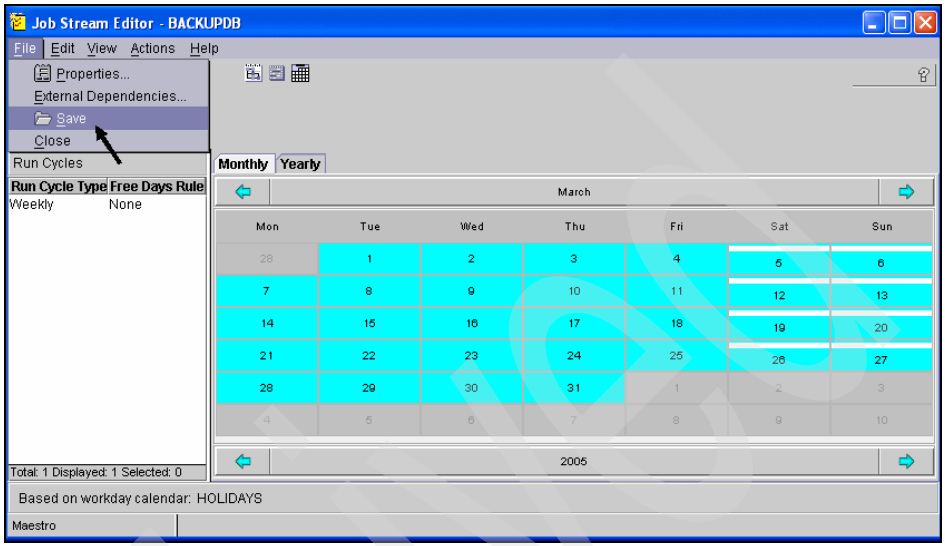

Figure 4-29 Dates selected by Run Cycle rule

14. Your job stream (Figure 4-30) is now complete and ready for execution after a Jnextday run, as it will be part of your production Symphony file. To submit the job stream immediately, use the **Submit** entry in the Action list or right-click your job stream in a job stream database list.

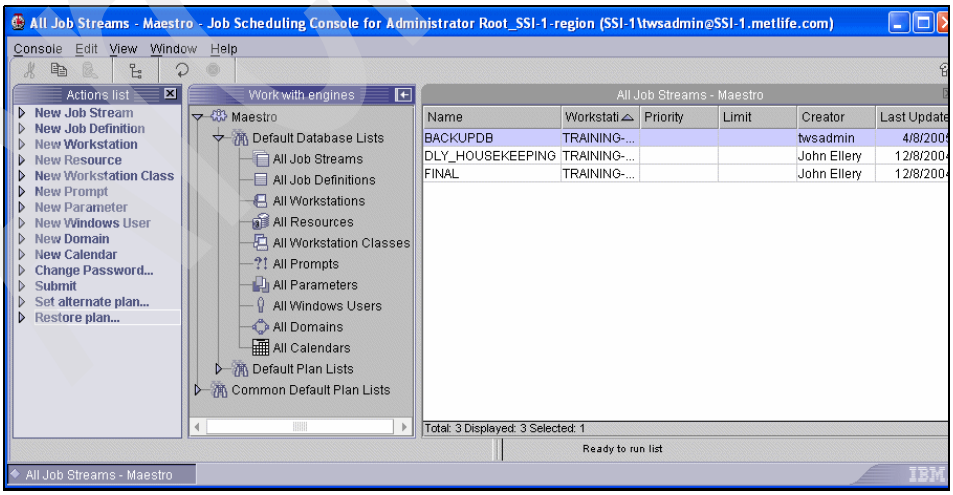

Figure 4-30 New job stream listed in All Job Streams list

Important: You will not be able to display the newly created schedule or job until a Jnextday run. The Jnextday script processes the following steps:

1. Run **`schedulr`** to select the appropriate schedules for the new day.
2. Run **`compiler`** to create an interim Production Control file.
3. Run **`reptr`** to print the preproduction reports.
4. Stop TWS processing.
5. Run **`stageman`** to carry forward uncompleted schedules, log the old Production Control file, and install the new file.
6. Start TWS processing for the new day.
7. Run **`reptr`** to print the post-production reports from the most recent log file.
8. Run **`logman`** to log job statistics from the most recent log file.

9. When a Jnextday is completed, choose the appropriate Plan List and display the production environment. Depending on security, this may not be possible. Ask your TWS administrator for your security options.

   Choose the **All Scheduled Job Streams** plan list to display the job stream name, state of the stream state, its priority, the schedule's duration, the number of jobs in that schedule, and the schedule's limit and other information about the streams. Use the scroll bar under the listed jobs to see additional information about the job streams.

   In our example, the BACKUPDB job stream is shown in the Successful state (internal status of **`SUCC`**), with a priority of 10, number of jobs, and other pertinent information (Figure 4-31).

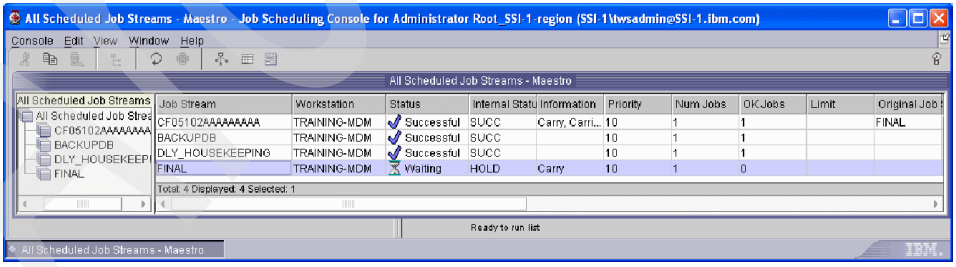

Figure 4-31 BACKUPDB Job Stream listing

10. Choose the **All Scheduled Jobs** plan list to display the job name, job number, job workstation, job stream, job stream workstation, status, internal

state, and other information about the job. Use the scroll bar under the listed jobs to see additional information about the job streams.

The example display in Figure 4-32 shows the BACKUP_DATABASE job's status is Successful (internal status of SUCC), the job number was 1036, the job is in the BACKUPDB stream, and other relevant information.

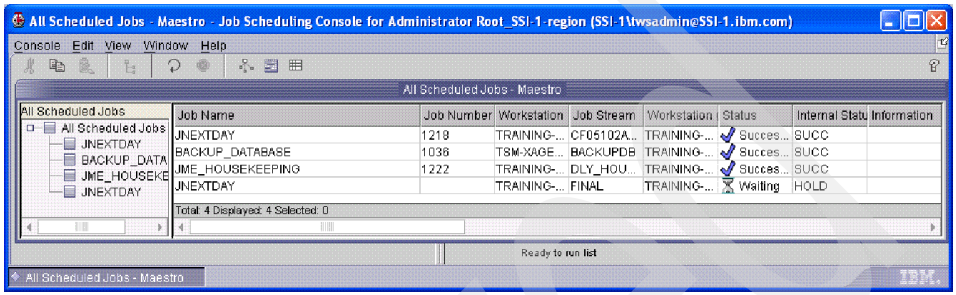

Figure 4-32 BACKUP_DATABASE job listing

11. Right-click the BACKUP_DATABASE job and select **Browse Job Log** to retrieve the log file that was created during job execution (Figure 4-33).

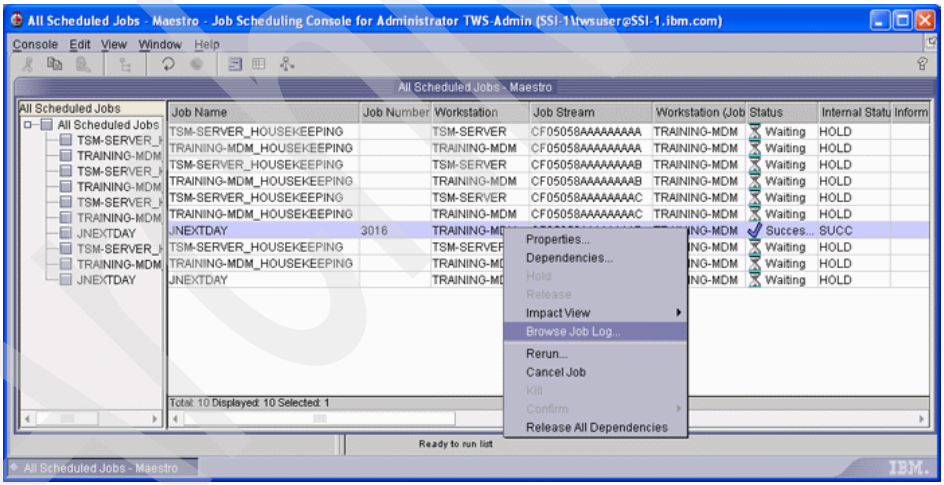

Figure 4-33 Checking the status of a job

12. The job log (also known in TWS as the stdlist) displays the header information: logon user, the job ID (or process ID), and the start date and time (Figure 4-34). The body of the stdlist contains the standard output of the job and its exit code, which determine the status of the job.

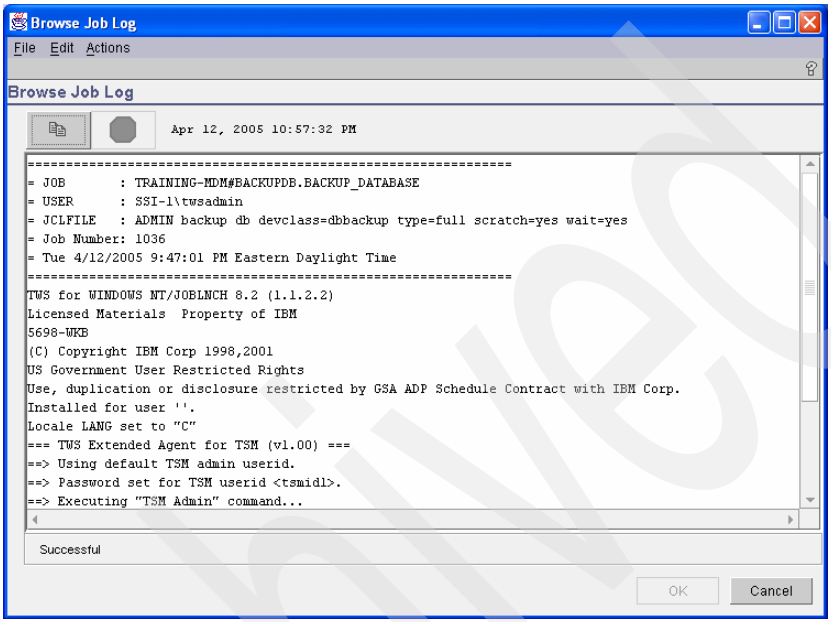

Figure 4-34 Job log for BACKUPDB

Follow the same steps when scheduling any job or job stream. The dependencies are the only differences.

Here is a summary of the steps for scheduling a database-backup job in TWS. To back up a database, you must define a copy of a storage pool on your server. Eliminating this copy results in errors when scheduling in TWS. To define a copy of a storage pool on your server, refer to the relevant user guide for your server platform.

1. From the TWS Job Scheduling Console, click the appropriate job type from the appropriate Scheduling Engine associated with the New Job Definition entry in the Action list.

Archived

2. Following the example in Figure 4-35, select the corresponding workstation and enter the job name (following your naming conventions).
3. Enter the logon ID. This is the user ID that will execute the job.
4. Enter the Recovery Option. If the job does not complete successfully, you must choose an action. The Stop option will wait for your manual next step. The Continue option would not be used in this scenario. The Rerun option enables the job to be executed again without manual intervention.

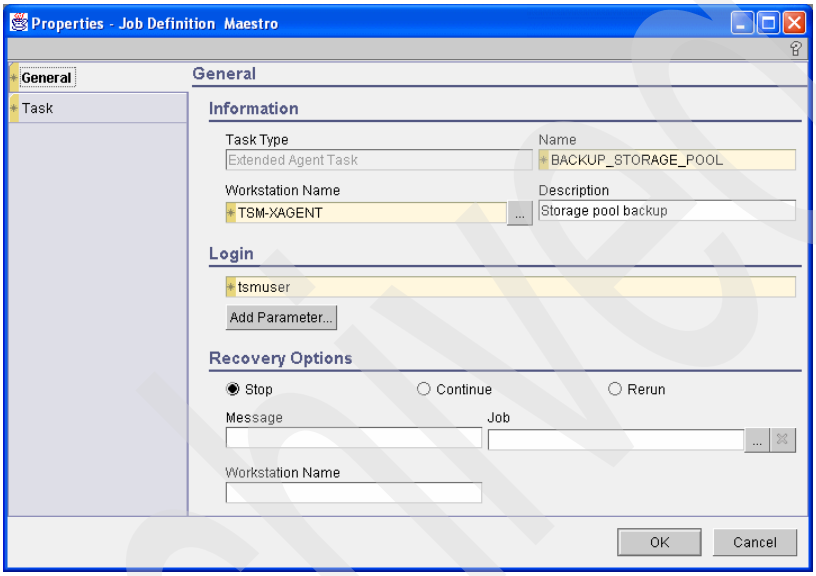

Figure 4-35 Job Definition General tab for storage pool backup

Archived

5. On the Task tab (Figure 4-36), enter the command or script (including path). The path can be entered in a parameter, in the .jobmanrc file, or in the command or script field.
6. Specify return code mapping if the command or script does not follow the convention of 0 for success and any other value is an abnormal end. Click **OK** to save changes and close the Properties window.

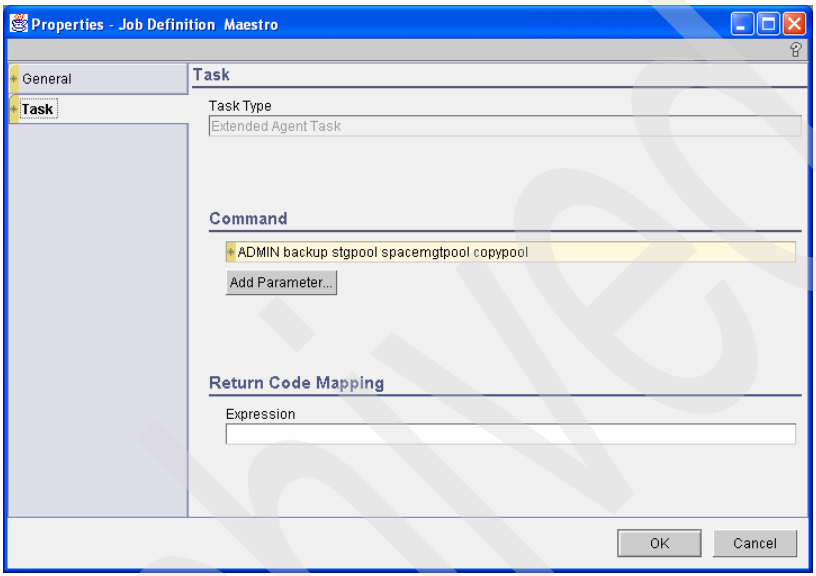

Figure 4-36 Job Definition Task tab for storage pool backup

7. To define a new job stream for this job, open the **New Job Stream** entry in the Action list and select the appropriate scheduling engine name.
8. Choose the associated workstation name and enter the job stream name.
9. Define a run cycle to specify when to execute the job stream.
10. Define the job stream dependencies (if any).
11. Select **File** → **Save**, then **File** → **Close**.

    You now have a job and job stream in the TWS database to back up a storage pool. There are several ways to execute your job stream.

    a. If you have added this job stream and job in a production environment, you can only submit this job stream in an ad hoc fashion. Submit the job or job stream from the Submit entry in the Action list, choosing **Job Stream** or **Job**, choosing the appropriate Scheduling Engine name, and answering the necessary prompt.

Archive

b. If the job stream and job were created in a test environment, you can rerun the Jnextday script from the command line. This enables the job stream and job to run in the Symphony file (or TWS production day). The job stream will execute immediately (if no time was entered) or it will execute on the given time.

> **Note:** A new run number is generated when a Jnextday process is executed and all executing jobs will appear as CF (carryforward job streams.)

To view the state of the job and job streams:

12. Select a job or job stream plan list from the from the Work with engines pane. You can select the default **All Scheduled Jobs** and **All Scheduled Job Streams** lists or define your own lists to display only the jobs and job streams of interest.
13. Use the Explorer view associated with the Job Stream lists or Job lists to view the state of the job.
14. To display the standard output or result of the job, right-click the mouse button and choose **Browse Job Log.**

Archived

4.6.2 Backup device configuration

You can use this command to back up information about device class definitions, library definitions, and drive definitions. If you do not have a library installed on your server, you can also back up information about your disk.

To schedule backup device configuration using TWS, you must define a new job stream and job. Follow the steps described in 4.6.1, “Database backup” on page 54 to prepare your job and job stream.

Figure 4-37 shows the Job Definition Task tab for backup device configuration.

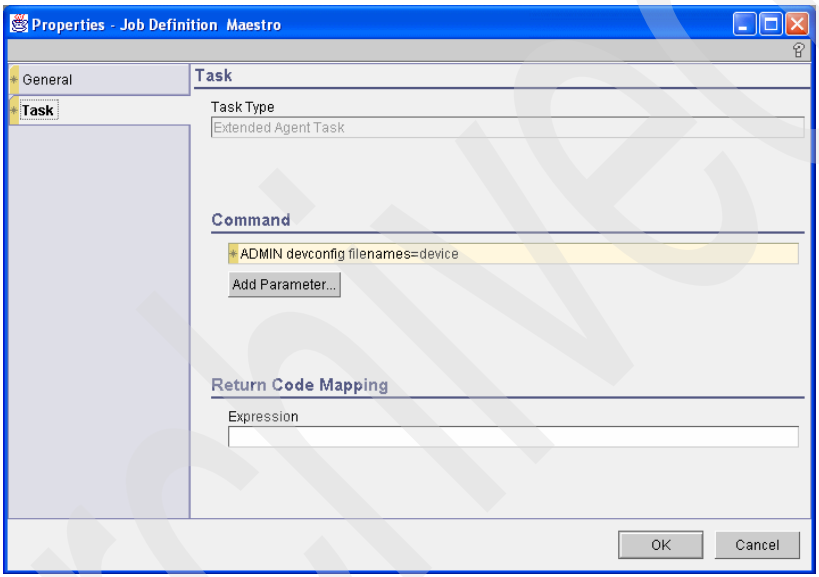

Figure 4-37 Job Definition Task tab for backup device configuration

15. After you create the new schedule and job to back up a device configuration, test your newly created schedule and job as in 4.6.1, “Database backup” on page 54.

4.6.3 Backup volume history

You can use this command to back up sequential volumes to one or more files.

To define a new job and a new schedule for this event, follow the steps described in 4.6.1, “Database backup” on page 54.

Figure 4-38 on page 75 shows the Job Definition Task tab for backup volume history.

Archived

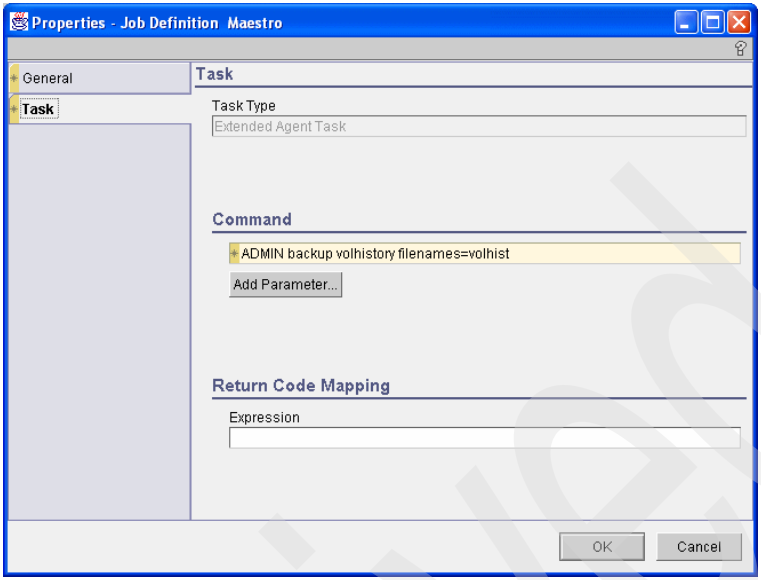

Figure 4-38 Job Definition Task tab for backup volume history

4.6.4 Clean volume history

Use this command to delete volume history file records that are no longer needed (that is, obsolete databases). To define a new job and a new schedule for this event, follow the steps described in 4.6.1, “Database backup” on page 54. Figure 4-39 shows the Job Definition Task tab for clean volume history.

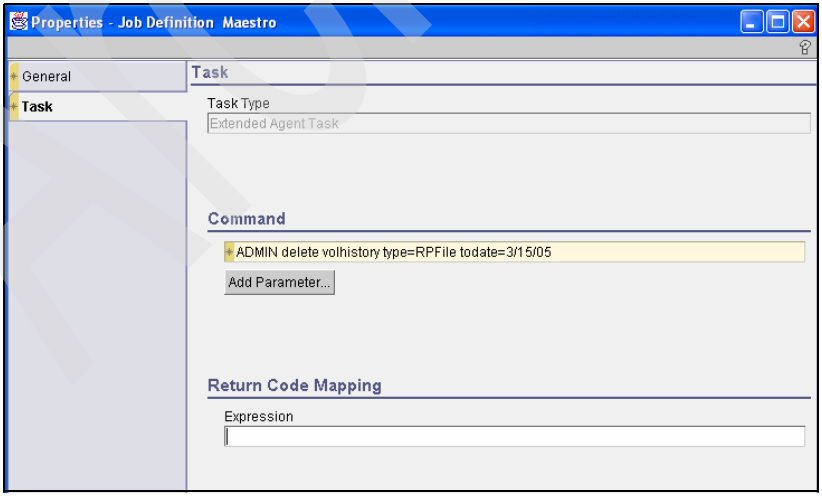

Figure 4-39 Job Definition Task tab for clean volume history

Archived

4.6.5 Expiration process

This inventory expiration process removes client backup and archive file copies from server storage based on policy specified in the backup and archive copy groups of the management classes to which the files are bound.

To define a new job and a new schedule for this event, follow the steps described in 4.6.1, “Database backup” on page 54.

Figure 4-40 shows the Job Definition Task tab for the expiration process.

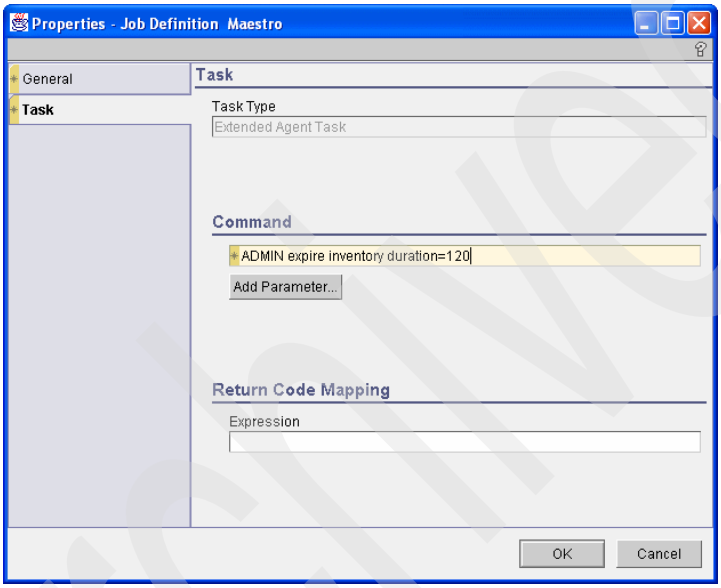

Figure 4-40 Job Definition Task tab for the expiration process

4.6.6 Reclamation process

Reclamation makes the fragmented space on volumes usable again by moving any remaining active files from one volume to another volume, thus making the original volume available for reuse.

To define a new job and a new schedule for this event, follow the steps described in 4.6.1, “Database backup” on page 54.

Figure 4-41 on page 77 shows the Job Definition Task tab for the reclamation process.

Archived

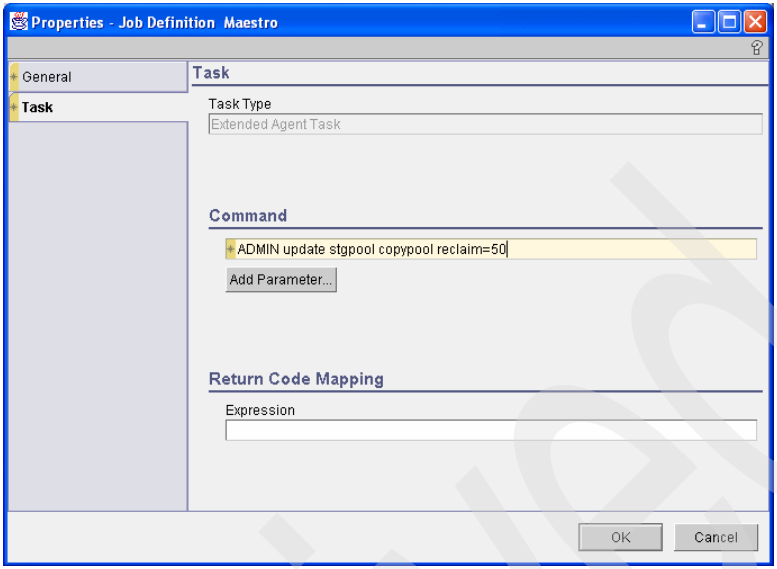

Figure 4-41 Job Definition Task tab for the reclamation process

4.6.7 Migration process

You can use a primary storage pool as the destination for backup files, archive files, or files migrated from client nodes. To define a new job and a new schedule for this event, follow the steps described in 4.6.1, “Database backup” on page 54. Figure 4-42 shows the Job Definition Task tab for the migration process.

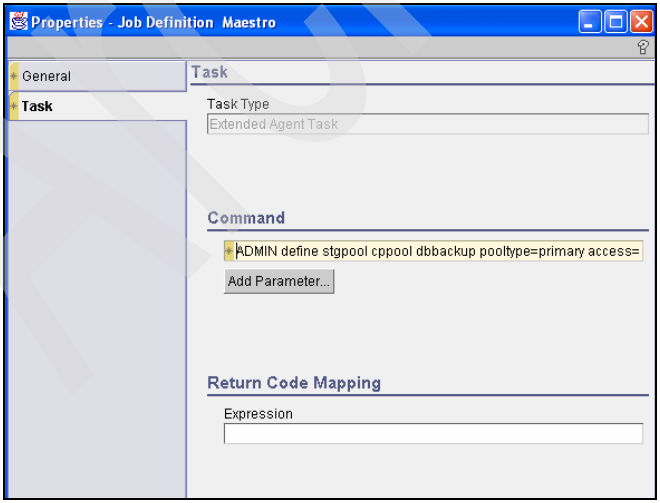

Figure 4-42 Job Definition Task tab for the migration process

Archive

4.6.8 Restore

You can restore files, a single database, databases, or volume history files. For more information, see the *Tivoli Storage Manager for AIX Administrator's Reference,* GC32-0768. To define a new job and a new schedule for this event, follow the steps described in 4.6.1, "Database backup" on page 54.

Figure 4-43 shows the Job Definition Task tab for restore.

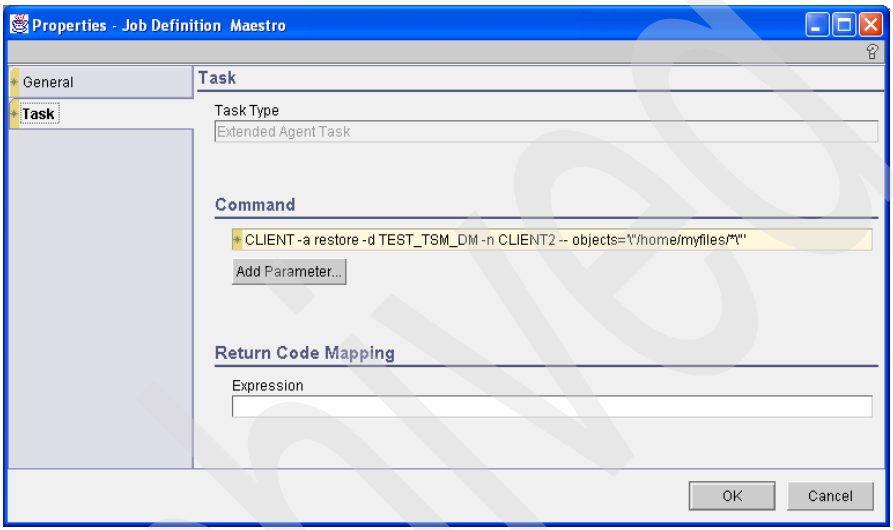

Figure 4-43 Job Definition Task tab for backupset restore

4.7 Summary

In this chapter, we investigated the following TSM functions that can be scheduled by using TWS extended agent for TSM.

- Backup device configuration
- Backup volume history
- Clean volume history
- Database backup
- Expiration process
- Migration process
- Reclamation process
- Restore

Instead of using the native TSM scheduler, we used our Tivoli Workload Scheduler custom extended agent for TSM to schedule these tasks through TWS. This way we were able to use TWS extended scheduling capabilities.

Archived

A

TSM Extended Agent source code

This appendix provides the Tivoli Storage Manager Extended Agent code, the Parms script, and tsmxagent.opts file. Note that these are provided on an as-is basis and can be downloaded from the ITSO Web site. For download instructions, refer to Appendix B, “Additional material” on page 101.

Important: The TSM Extended Agent script runs in Korn Shell environment. We have tested the script on AIX, but it should run with little or no modification on any platform that supports Korn Shell.

© Copyright IBM Corp. 2005. All rights reserved.

Parms script code

Example A-1 is the Parms script to be used with TSM Extended Agent.

Example A-1 Parms script

```
#!/bin/sh
if [ "$2" ]
then
   encrypt=`echo $2 | tr
0123456789AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
HtGuFvEwDxCyBzAaZ0bY1cX2dW3eV4fU5gT6hS7iR8jQ9kPlOmNnMoLpKqJrIs`
   parms -c $1 $encrypt
else
   value=`parms $1 | tr
HtGuFvEwDxCyBzAaZ0bY1cX2dW3eV4fU5gT6hS7iR8jQ9kPlOmNnMoLpKqJrIs
0123456789AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz`
   echo "$value"
fi
```

TSM Extended Agent code

Example A-2 shows the TSM Extended Agent code.

Example A-2 TSM Extended Agent source code

```
#!/bin/ksh
set +u
#-------------------------------------------------------------------------
# This method script executes commands on the Tivoli Storage Manager server
# for administrative and client backup purposes
#
#***********
# Job scheduling syntax:
#***********
#
# Create jobs for this x-agent with the following commands in the
# "script name" field on the job definition:
############
#
# ADMIN <TSM Admin Command with options>
#
# When entering a double quote " in a TSM Admin Command, be sure to preceed it
# with a \ (ex. \").  This is required because the TWS JSC uses double quotes
# as input delimiters.
#
```

Archived

```
# WARNING!: some TSM Admin commands require the WAIT=YES option or they will
#           schedule an action and immediately return a return code of 0.
#
#************
# ADMIN Example 1. Create a backup
#************
# ADMIN generate backupset cbuehler2 bkpset dev=TSM_test_tape ret=10 wait=yes
#************
############
#
# CLIENT -a <action> -d <policy domain> -n <TSM node name>
#        [-p value] [-s <value> [-t <value> ] [-- <TSM options>]
#
#
#  -a <action>TSM Define Schedule action (see TSM Admin Ref)
#  -d <policy domain>TSM Policy Domain
#  -p <polling interval>Time to wait before the next status poll
#  -n <TSM node name>Name of the TSM client to execute the script on
#  -s <script path\\name>The script to exec when the "-a command" is
#                              specified.  Use '\\' instead of '\'
#            WARNING!: due to script name parsing, avoid
#                      using " -" in the script path\name.
#  -t <timeout count>Number of polls to complete before timing out.
#            Defaults to 0 = Poll Forever
#  -w        Execute the client command synchronously
#            Commands will not timeout so the "-p" & "-t"
#            options are ignored.
#  -- <TSM options>       Any TSM options you need to add to the
#                              TSM Define Schedule command to be executed.
#            Note: when multiple values are required for
#              a TSM option, enclose the values in
#              escaped double quotes.
#              ex. \"<values>\"
#
#  -a, -d, & -n options are REQUIRED!
#
#  WARNING!: If you use the "--" option it MUST be the LAST option in
#       the job command string.
#************
# CLIENT Example 1a. Execute a script asynchronously (Windows)
#************
# CLIENT -a command -d TSM_TEST_DM -p 30 -s C:\\Program
Files\\Tivoli\\TSM\\SCRIPTS\\tsm_rc_0.bat -n cbuehler1
#
# This will run a script on a TSM client via the TSM scheduler.
# Note the requirement for using 2 backslashes instead of single backslashes
# in the script path\name.  This is necessary due to variable substitution
```

Archived

```
# by TWS and the TWS extended agent (this script).
#************
# CLIENT Example 1b. Execute a script asynchronously (Unix)
#************
# CLIENT -a command -d TSM_TEST_DM -w -s /home/maestro/script_name -n cbuehler1
#************
# CLIENT Example 2. Incremental backup with TSM options
#************
# CLIENT -a incremental -d TSM_TEST_DM -p 600 -n cbuehler1 --
OPTIONS=\"-nojournal -optfile=SQLLOG.opt\"
#
# This will perform an incremental backup of the client with the "-nojournal"
# and "-optfile" options set.  Please note that the "s are escaped (\").  This
# is required because the TWS code that processes the job command field uses
# "s as delimiters.
#************
# Installation:
# > Install a TWS FTA on the AIX TSM Server
# > Use the TWS `dumpsec` & `makesec` commands to add the AIX uid that
#   XA jobs will be executed with to the TWS security file.
# > Copy this script into the TWS FTA 'methods' on the AIX TSM Server and
#   set the file permissions to 750.
# > Copy the "PARMS" script to the 'methods' directory as well.
# > Using the JSC, create a "New Workstation" definition for the XA on the
#   TSM Server.
#
#   Notes:
#   1. When creating the "New Workstation", be certain to specify
#  OS = Other
#  Workstation Type = Extended Agent
#   2. Until Jnextday is run on the TWS server, the XA is not usable in a plan
#
# > Verify that the default value for "USERID" in this script is correct.
#   This variable contains the TSM userid that is used to execute TSM cmds.
# > In the TWS FTA 'methods' directory on the AIX TSM Server, execute the TWS
#   `PARMS <TSM_userid> <TSM_password>` command to set the TSM password used
#   for each TSM uid that will be used to schedule TSM commands via the XA
# > Test the installation by scheduling a TSM job to run on the XA
#************
# Implementation Notes:
# > The TWS `parms` command is used to store & retrieve the encrypted TSM pw.
# > The 'parameters' file in the TWS install directory is used to store
#   the encrypted TSM password.
# > The 'parameters.key' file is required to decrypt the values stored using
#   the TWS `parms` utility.
# > TWS interprets all return codes returned by fail_job & abend_job as
#   modulus 256 values.  The correct job completion state (ABEND) is displayed
#   as expected and the job log text indicates the actual return code.
#   Examples...
```

```
#  job rc of 256 =   0 in TWS job summary info
#  job rc of 257 =   1 in TWS job summary info
#  job rc of 999 = 231 in TWS job summary info
# > You can override the TSM admin userid specified in the USERID variable
#   by creating a file called "tsmxagent.opts" in the home directory of the
#   UNIX uid used to execute the job.  This file should contain the
following...
#   tsmAdmin <TSM_Admin_userid>
#
#   where <TSM_Admin_userid> is the TSM Admin userid you want put in USERID
#************
# Change Control:

#--------------------------------------------------------------------------

# Set DEBUG flag to a value other than 0 to activate debug messages
DEBUG=0

############
# Initialize Variables
############

PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/etc/:/posix:/usr/ucb:/usr/bsd
export PATH

# Initialize some of the variables.
SCHEDULE_NAME=""
FORMATTED_SCHED_DATE=""
UNIVERSAL_DATE=""
NODE_NAME=""
JOB_NAME=""
JOB_ID=""
STREAM_LOGON=""
MAESTRO_CPU=""
MAESTRO_HOST=""
MAESTRO_MASTER=""
PORT_NUMBER=""
TEST_OPTIONS=""
CURRENT_RUN_NUMBER=""
REQUIRED_RUN_NUMBER=""
STDLIST=""
TASK=""
PARENT="NO"
XARG1=""
XARG2=""
XARG3=""
TMPJOBNAME=""
JOBSTATUS=""
DEVCLASS=""
```

Archived

```
BACKUPTYPE=""
FILENAMES=""
TODATE=""
DURATION=""
RET9=""
DSMADMC="/usr/bin/dsmadmc"

# Set the standard banner message.
BANNER='=== TWS Extended Agent for TSM (v1.00) ==='

# Set the standard usage message.
USAGE="Invalid Usage. See Tivoli Workload Scheduler Users Guide for correct
usage ."

# Set the TSM Client action options usage message
CLIENT_USAGE=": option is NOT valid.  Valid options are...\n\t-a <action>\n\t-d
<policy domain>\n\t-p <status polling interval (seconds)>\n\t-t <timeout count
(number of polls)>\n\t-s <script path\\\\\\\\name>\n\t-- <TSM options>"

# Set the default number of seconds between TSM schedule status checks
CHECKFREQ=60

# Set the default number of TSM Schedule status checks before
# the job times out and the schedule is deleted
# Note: 0 = poll forever
JOBTIMEOUT=0

# Set default TSM userid
USERID="tsmid1"

# Set the XA path and program names
if [ -z "$UNISON_EXEC_PATH" ]
 then
    PROG_NAME=`basename $0`
    EXEC_PATH=`dirname $0`
 else
    PROG_NAME=`basename $UNISON_EXEC_PATH`
    EXEC_PATH=`dirname $UNISON_EXEC_PATH`
fi

# Set the maestro home directory path
MAE_HOME=`dirname $EXEC_PATH`

# Add maestro home to PATH.
PATH=$MAE_HOME/bin:$MAE_HOME:$PATH
export PATH

# Set the path to the PARMS encryption script
PARMS=$EXEC_PATH"/PARMS"
```

Archived

```
########
# Define Functions
########

echoerr()
# Function to display error messages.
{
   TIME=`date +%H:%M`
   echo "${PROG_NAME}:$TIME/" "$@" >&2

   # Display the error message in the job log
   echo "ERROR: $@"
}

print_version()
{
    echo "$BANNER"
}

command_exist()
{

    if type "$@" 2>&1 | grep -i "not *found" > /dev/null 2>&1
     then
       echoerr "Command $@ not found."
       return 1
     else
       return 0
    fi
}

Check_File()
# Check for existance of a file on the extended agent machine.
{
   [ -z "$TEST_OPTIONS" ] && TEST_OPTIONS="-f"
   test $TEST_OPTIONS "$FILE_NAME"
   exit $?
}

fail_job()
{
   # If an attempt has not already been made to delete the schedule...
   if [[ -z $del_attempted ]]
   then
      # Indicate that an attempt is being made to delete the schedule
      del_attempted=1

      # Attempt to delete the schedule
```

Archived

```
      tsm_del_schedule
   fi

   send_Message "CJ FAIL"
   echoerr ">>> Job FAIL <<<"
   exit $1
}

abend_job()
{
   # If an attempt has not already been made to delete the schedule...
   if [[ -z $del_attempted ]]
   then
      # Indicate that an attempt is being made to delete the schedule
      del_attempted=1

      # Attempt to delete the schedule
      tsm_del_schedule
   fi

   send_Message "CJ ABEND"
   echoerr ">>> Job ABEND <<<"
   exit $1
}

succeed_job()
{
   # If an attempt has not already been made to delete the schedule...
   if [[ -z $del_attempted ]]
   then
      # Indicate that an attempt is being made to delete the schedule
      del_attempted=1

      # Attempt to delete the schedule
      tsm_del_schedule
   fi

   send_Message "JS"
   echo ">>> Job SUCC <<<"
   exit 0
}


# Run dsmserv admin command
{

   # Save the TSM Admin command options
   TSM_OPTIONS=$*
```

Archived

```
   # Remove all \"s
   TSM_OPTIONS=$(echo $TSM_OPTIONS | sed 's/\\"/\"/g')

   # Create command to define the schedule to run the user script
   cmd="$DSMADMC -id=$USERID -password=$PASSWORD $TSM_OPTIONS"

   # Execute the TSM Admin command
   echo "==> Executing \"TSM Admin\" command..."
   tsm_exec_cmd

   # The job completed successfully
   succeed_job

}


tsm_client_command()
# run TSM Client action via tsm scheduler
{

   # Parse the script options
   while [ $# -gt 0 ]
   do
      case $1 in
        -a)
           OPT_ARG="$1"
           shift
           # Set TSM Define Schedule Client action
           TSM_ACTION=$1
           shift
          ;;
        -d)
           OPT_ARG="$1"
           shift
           # Set TSM policy domain
           TSM_POLICY_DOMAIN=$1
           shift
          ;;
        -n)
           OPT_ARG="$1"
           shift
           # Set TSM target client node name
           TSM_NODE_NAME=$1
           shift
          ;;
        -p)
           OPT_ARG="$1"
           shift
           # Set TSM status polling interval in seconds
```

Archived

```
    CHECKFREQ=$1
    shift
   ;;
  -s)
    OPT_ARG="$1"

# Set the user script path/name
while [ $# -gt 0 ]
    do
      shift
  # If a "-" is encountered, the script path/name is complete
  # and the next option needs to be processed
  [[ `echo "$1" | cut -c 1` = "-" ]] && break
      SCRIPT=$SCRIPT" $1"
    done

# Remove the space at the beginning of the script path/name
    SCRIPT=`echo $SCRIPT | cut -c 1-`

# Add the "objects=" TSM Define Schedule option tag and
# enclose the script string in both single & double quotes
SCRIPT="objects='\""$SCRIPT"\"'"
   ;;
  -t)
    OPT_ARG="$1"
    shift
    # Set TSM command timeout count (# of polling intervals)
    JOBTIMEOUT=$1
    shift
   ;;
  -w)
    OPT_ARG="$1"
    shift
    # Set the flag indicating that the "define clientaction" command
# should be used to wait for the command to execute synchronously.
    TSM_WAIT=1
   ;;
  --)
# Skip the "--" option tag
shift

# Any remaining options should be TSM Options because this option
# is required to be last by the CLIENT command syntax implemented
    # in this XA (script)
TSM_OPTIONS=$*

TSM_OPTIONS=$(echo $TSM_OPTIONS | sed 's/\\//g')

# There are no more options to process so exit the option loop
```

Archived

```
      break
          ;;
        *)
          OPT_ARG="$1"
          echoerr "\"$OPT_ARG\"$CLIENT_USAGE"
          shift

      # Indicate that a syntax error has been encountered so the XA
      # will exit without executing a command.
          syntax_error=1
         ;;
       esac
   done

   # Ensure that required parameters are specified...
   if [[ -z $TSM_ACTION ]]
   then
      echoerr "\"-a\" <action> is a required parameter."
      syntax_error=1
   fi
   if [[ -z $TSM_POLICY_DOMAIN ]]
   then
      echoerr "\"-d\" <policy domain> is a required parameter."
      syntax_error=1
   fi

# If the workstation name starts with "xa_" then override the TSM node name
   if [[ -n `echo $MAESTRO_CPU | grep -i "^xa_"` ]]
   then
      TSM_NODE_NAME=${MAESTRO_CPU##*_}
      echo "==> Overriding -n <TSM_NODE_NAME> with $TSM_NODE_NAME based on
CPU=$MAESTRO_CPU"
   fi

   if [[ -z $TSM_NODE_NAME ]]
   then
      echoerr "\"-n\" <TSM node name> is a required parameter."
      syntax_error=1
   fi
   if [[ "$TSM_ACTION" = @(command|COMMAND) && -z $SCRIPT ]]
   then
      echoerr "\"-s\" <script path\\\\\\\\name> is a required parameter when
the \"-a command\" action is specified."
      syntax_error=1
   fi
   if [ $syntax_error ]
   then
      fail_job 1
   fi
```

Archived

```
   # Tell TWS to indicate that the job is in the "EXEC" state
   send_Message "CJ EXEC"


   # If the "execute the client command synchronously" flag is set...
   if [[ $TSM_WAIT = 1 ]]
   then
      # Execute the command synchronously
      tsm_clientaction
   else
      # Schedule the command for asynchronous execution
      tsm_client_schedule
   fi

}

tsm_client_schedule()
{

   # Set the temporary job name for the client script job
   TSM_SCHEDULE_NAME="TWSJOB_"$JOB_ID

   # Create command to define the schedule to run the user script
   cmd="$DSMADMC -id=$USERID -password=$PASSWORD define schedule
$TSM_POLICY_DOMAIN $TSM_SCHEDULE_NAME action=$TSM_ACTION startdate=today
starttime=now dayofweek=any expiration=TODAY $SCRIPT $TSM_OPTIONS"

   # Execute the "define schedule" command
   echo "==> Executing \"define schedule\" command..."
   tsm_exec_cmd

   # Create the command to associate the user script with the schedule
   cmd="$DSMADMC -id=$USERID -password=$PASSWORD define association
$TSM_POLICY_DOMAIN $TSM_SCHEDULE_NAME $TSM_NODE_NAME"

   # Execute the "associate schedule" command
   echo "==> Executing \"associate schedule\" command..."
   tsm_exec_cmd

   # Poll for the schedule status
   tsm_poll_schedule

}

tsm_clientaction()
{

   # Create command to define the schedule to run the user script
```

Archived

```
   cmd="$DSMADMC -id=$USERID -password=$PASSWORD define clientaction
$TSM_NODE_NAME domain=$TSM_POLICY_DOMAIN action=$TSM_ACTION $SCRIPT
$TSM_OPTIONS wait=yes"

   # Execute the "define schedule" command
   echo "==> Executing \"define clientaction\" command..."
   tsm_exec_cmd

   # The job completed successfully
   succeed_job
}


tsm_del_schedule()
# Delete a TSM Schedule
{

   # If a TSM Schedule was created...
   if [[ -n $TSM_SCHEDULE_NAME ]]
   then

   # Create command to delete the TSM schedule created by this XA
        cmd="$DSMADMC -id=$USERID -password=$PASSWORD delete schedule
$TSM_POLICY_DOMAIN $TSM_SCHEDULE_NAME"
   echo "==> Executing \"delete schedule\" command..."
   tsm_exec_cmd

   fi

}


tsm_exec_cmd()
# Execute a TSM command and handle any errors
{

   # Execute the TSM command
   result=`$cmd`

        # Remove the carriage returns from the TSM command results
        rc=$(echo $result | sed 's/\n//g')

   # Parse the TSM command return code
   rc=${rc##*Highest return code was }
   rc=${rc%.}

   # If the DEBUG flag is not 0...
   if [[ $DEBUG != 0 ]]
   then
```

Archived

```
      echo "DEBUG: TSM command/result...\n$cmd\n\n$result"
   fi

   # If the TSM command failed...
   if [[ ! $rc -eq 0 ]]
   then
      echoerr "TSM command FAILED...\n$result"

      fail_job $rc
   fi

}

tsm_poll_schedule()
# Poll the TSM schedule for status
{

   # Poll TSM for the schedule status
   if [[ $JOBTIMEOUT -eq 0 ]]
   then
      echo "==> Begin TSM schedule wait/poll cycle...  Wait=$CHECKFREQ seconds
& Timeout='poll forever'"
   else
      echo "==> Begin TSM schedule wait/poll cycle...  Wait=$CHECKFREQ seconds
& Timeout=$JOBTIMEOUT polls"
   fi

   while [[ $JOBTIMEOUT -eq 0 || $COUNTER -lt $JOBTIMEOUT ]]
   do
      sleep $CHECKFREQ

      let COUNTER=$COUNTER+1

      # Create the command to query the schedule status
      cmd="$DSMADMC -id=$USERID -password=$PASSWORD query event
$TSM_POLICY_DOMAIN $TSM_SCHEDULE_NAME Nodes=$TSM_NODE_NAME"
      tsm_exec_cmd

      # Remove the carriage returns from the "query event" results
      result=$(echo $result | sed 's/\n//g')

      # Remove the text from the beginning thru the last "- "
      JOBSTATUS=${result##*- }

      # Remove the text from the end thru " ANS8002I"
      JOBSTATUS=${JOBSTATUS%% ANS8002I*}

      # Display schedule status summary
      echo "--> TSM schedule status... $JOBSTATUS"
```

Archived

```
      # Get the status of the TSM schedule
      STATUS=${JOBSTATUS##* }

      # If the TSM schedule has completed...
      if [[ "$STATUS" != @(Future|Pending|Restarted|Started) ]]
      then
      # Create command to query for detailed schedule status
           cmd="$DSMADMC -id=$USERID -password=$PASSWORD query event
$TSM_POLICY_DOMAIN $TSM_SCHEDULE_NAME Nodes=$TSM_NODE_NAME Format=Detailed"
           echo "==> Executing \"query event Format=Detailed\" command..."
           tsm_exec_cmd

           # Remove the carriage returns from the "query event" results
           JOBSTATUS=$(echo $result | sed 's/\n//g')

           # Remove the text from the end thru " ANS8002"
           JOBSTATUS=${JOBSTATUS%% ANS8002I*}

           # Get the action exit code (rc) from the detailed TSM status
           action_rc=${JOBSTATUS##* }

      # If the action exit code (rc) is greater than 0...
           if [[ $action_rc -gt 0 ]]
           then
              echoerr ">>> TSM Client action exited with rc=$action_rc <<<"

              fail_job $action_rc
           fi

      # The job completed successfully
           succeed_job
      fi
   done

   echoerr ">>> Job TIMEOUT <<<"
   fail_job 1
}

Launch_Job()
# Launch the job
{
   if [ $# -lt 2 ]; then
     echoerr "Invalid job definition. Too few arguments!"
     fail_job 1
   fi
   [ -z "$SCRIPT_NAME" ] && exit 1
```

Archived

```
   touch "$STDLIST"
   TMP_PATH=/tmp

   case $1 in
   ADMIN|admin)
      # Skip over the "ADMIN|admin" value in the parms
      shift

      tsm_admin_command $@
      ;;
   CLIENT|client)
      # Skip over the "CLIENT|client" value in the parms
      shift

      tsm_client_command $@
      ;;
   TST|tst)
   # Execute tst()
   shift

   tst $@
   ;;
   *)
      echoerr "Invalid job definition. Must be ADMIN or CLIENT!"
      fail_job 1
   esac

   exit 0
}

Check_Connection()
#Check connection
{
    exit 0
}

Manage_Job()
{
    exit 1
}

send_Message()
{
    echo "%" "$@"
    return 0
}

tst() to test "displaying intermediate status"
tst()
```

Archived

```
{

   echo "Executing test using the TST() function of the TWS XA for TSM\n"

   echo "Set WAIT state..."
   echo "%CJ WAIT"
   sleep 10

   echo "Set DONE state..."
   echo "%CJ DONE"
   sleep 10

   echo "Set PEND state..."
   echo "%CJ PEND"
   sleep 10

   echo "Set FAIL state..."
   echo "%CJ FAIL"
   sleep 10
   exit 0



}

#######
# Begin Main Script
#######

# Display the TWS XA version banner
echo $BANNER


# The name of the TSM administrator logon is in the tsmxagent.opts file
opts_file=`pwd`"/${PROG_NAME}.opts"
if [ -f $opts_file ]
  then
    USERID=`awk '$1 ~ /tsmAdmin/ { print $2 }' $opts_file`
    echo "==> Setting TSM admin userid from $opts_file"
  else
    echo "==> Using default TSM admin userid."
fi

# Get the TSM password
if command_exist parms
then
  if command_exist $PARMS
  then
    PASSWORD=`$PARMS $USERID`
```

Archived

```
  else
    echoerr "Encryption routine was not found.  Attempting to use unencrypted
password."
    PASSWORD=`parms $USERID`
  fi

  if [[ -n "$PASSWORD" ]]
  then
    echo "==> Password set for TSM userid <$USERID>."
  else
    echoerr "Execute \`$PARMS $USERID <password>\` to set password for TSM
userid <$USERID>.\nIf you are not using encrypted passwords, execute \`parms -c
$USERID <password>\`\n*** Note: If setting the password does not work add the
<userid>, specified in the string \"= USER: <userid>\" at the top of this job
log, to the TWS Security file. ***"
    exit 3
  fi
else
  echoerr "Could not find the TWS parms utility."
  exit 3
fi

# Parse the arguments passed to the XA by TWS
while [ $# -gt 0 ]
do
  OPT="$1"
  [ "`echo %$OPT | cut -c2`" != "-" ] && break
  shift

  case $OPT in
      -c)
          OPT_ARG="$1"
          shift
          MAESTRO_CPU=`echo "$OPT_ARG" | cut -d',' -f1`
          MAESTRO_HOST=`echo "$OPT_ARG" | cut -d',' -f2`
          MAESTRO_MASTER=`echo "$OPT_ARG" | cut -d',' -f3`
        ;;
      -d)
          OPT_ARG="$1"
          shift
          FORMATTED_SCHED_DATE=`echo "$OPT_ARG" | cut -d',' -f1`
          UNIVERSAL_DATE=`echo "$OPT_ARG" | cut -d',' -f2`
        ;;
      -j)
          OPT_ARG="$1"
          shift
          JOB_NAME=`echo "$OPT_ARG" | cut -d',' -f1`
          JOB_ID=`echo "$OPT_ARG" | cut -d',' -f2`
        ;;
```

Archived

```
-l)
     OPT_ARG="$1"
     shift
     STREAM_LOGON="$OPT_ARG"
  ;;
-n)
     OPT_ARG="$1"
     shift
     NODE_NAME="$OPT_ARG"
  ;;
-o)
     OPT_ARG="$1"
     shift
     STDLIST="$OPT_ARG"
  ;;
-p)
     OPT_ARG="$1"
     shift
     PORT_NUMBER="$OPT_ARG"
  ;;
-q)
     OPT_ARG="$1"
     shift
     TEST_OPTIONS="$OPT_ARG"
  ;;
-r)
     OPT_ARG="$1"
     shift
     CURRENT_RUN_NUMBER=`echo "$OPT_ARG" | cut -d',' -f1`
     REQUIRED_RUN_NUMBER=`echo "$OPT_ARG" | cut -d',' -f2`
  ;;
-s)
     OPT_ARG="$1"
     shift
     SCHEDULE_NAME="$OPT_ARG"
  ;;
-t)
     OPT_ARG="$1"
     shift
     TASK="$OPT_ARG"
  ;;
-V)
     print_version
     exit 0
  ;;
--)
    break
  ;;
 *)
```

Archived

```
            echoerr "$USAGE"
            exit 1
          ;;
      esac
done


# There could be arguments that were not parsed above so
# save them...
REMAINING_ARGS="$*"


if [ ! "$TASK" = "CC" ]
then
  if [ "$#" = "0" ] || [ -z "$TASK" ]
  then
    echoerr "$USAGE"
    exit 1
  fi
fi


# Take action based on the value of the "-t <TASK>" passed by TWS
case $TASK in
  CF)
       FILE_NAME="$REMAINING_ARGS"
       Check_File
    ;;
  MJ)
       JOB_PID=`echo "$REMAINING_ARGS" | cut -d" " -f1`
       STDLIST=`echo "$REMAINING_ARGS" | cut -d" " -f2`
       Manage_Job
    ;;
  LJ)
       SCRIPT_NAME="$REMAINING_ARGS"
       send_Message "CJ WAIT"
       Launch_Job $SCRIPT_NAME
    ;;
  CC)
       Check_Connection
    ;;
   *)
     echoerr $USAGE
     send_Message "UT $USAGE"
     exit 2
    ;;
esac

# If the script "fall's thru" to here, something went wrong
```

```
# so exit with an error
echoerr "Unexpected error.  Contact TWS XA for TSM owner."
abend_job 1
```

Sample tsmxagent.opts file

Example A-3 shows the sample tsmxagent.opts file.

Example A-3 Sample tsmzagent.opts file

```
tsmAdmin tsmid2
```

Archived

Archived

B

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/SG246696

Alternatively, you can go to the IBM Redbooks Web site at:

http://www.redbooks.ibm.com

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246696.

Using the Web material

The additional Web material that accompanies this redbook includes this file:

File name	*Description*
SG246696.zip	Zipped TSM Extended Agent code

© Copyright IBM Corp. 2005. All rights reserved.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 30 MB minimum
Operating system: Windows/Linux/AIX

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.

Archived

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 this publication, see How to get IBM Redbooks.

- *IBM Tivoli Workload Scheduler Version 8.2: New Features and Best Practices*, SG24-6628

Other publications

These publications are also relevant as further information sources:

- *IBM Tivoli Storage Manager for AIX Administrator's Reference V5.3*, GC32-0769
- *IBM Tivoli Workload Scheduler 8.2 Planning and Installation Guide,* SC32-1273
- *IBM Tivoli Workload Scheduler 8.2 Reference Guide*, SC32-1274
- *IBM Tivoli Workload Scheduler Job Scheduling Console 1.3 User's Guide*, GC32-1257

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:

`http://www.redbooks.ibm.com`

Help from IBM

IBM Support and downloads

`ibm.com/support`

Archived

© Copyright IBM Corp. 2005. All rights reserved.

IBM Global Services

ibm.com/services

Archived

Index

Archived

© Copyright IBM Corp. 2005. All rights reserved.

Archived

T

U

Archived

Archived

REDBOOKS

(0.2”spine)
0.17”<->0.473”
90<->249 pages

Archived

Archived

Implementing IBM Tivoli Workload Scheduler V8.2 Extended Agent for IBM Tivoli Storage Manager

Insider's guide to Tivoli Workload Scheduler extended agents

Ready-to-use solution for TSM and TWS integration

TSM Extended Agent code included

IBM Tivoli Workload Scheduler is a strategic, multiplatform distributed scheduling product that provides high-volume, complex scheduling capability. Although Tivoli Workload Scheduler provides native support for many platforms and applications, its robust scheduling capabilities can be extended to cover additional platforms and applications by writing an extended agent.

This IBM Redbook shows how to write a Tivoli Workload Scheduler Version 8.2 extended agent to schedule jobs on IBM Tivoli Storage Manager. With the extended agent, you can schedule on platforms and applications for which Tivoli Workload Scheduler has no native agent, as well as integrate IBM Tivoli Storage Manager with Tivoli Workload Scheduler. The Tivoli Workload Scheduler scheduling facility enables you to assign dependencies among tasks scheduled through Tivoli Storage Manager or to assign limits or priorities. By extending Tivoli Storage Manager to schedule these Tivoli Storage Manager tasks, you can take advantage of its advanced scheduling capabilities.

This book will be essential for those who write a Tivoli Workload Scheduler extended agent for any platform in general, or use the extended agent provided in this book (TSM Extended Agent) to schedule Tivoli Storage Manager tasks.

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

SG24-6696-00 ISBN 0738491063