Tivoli. software

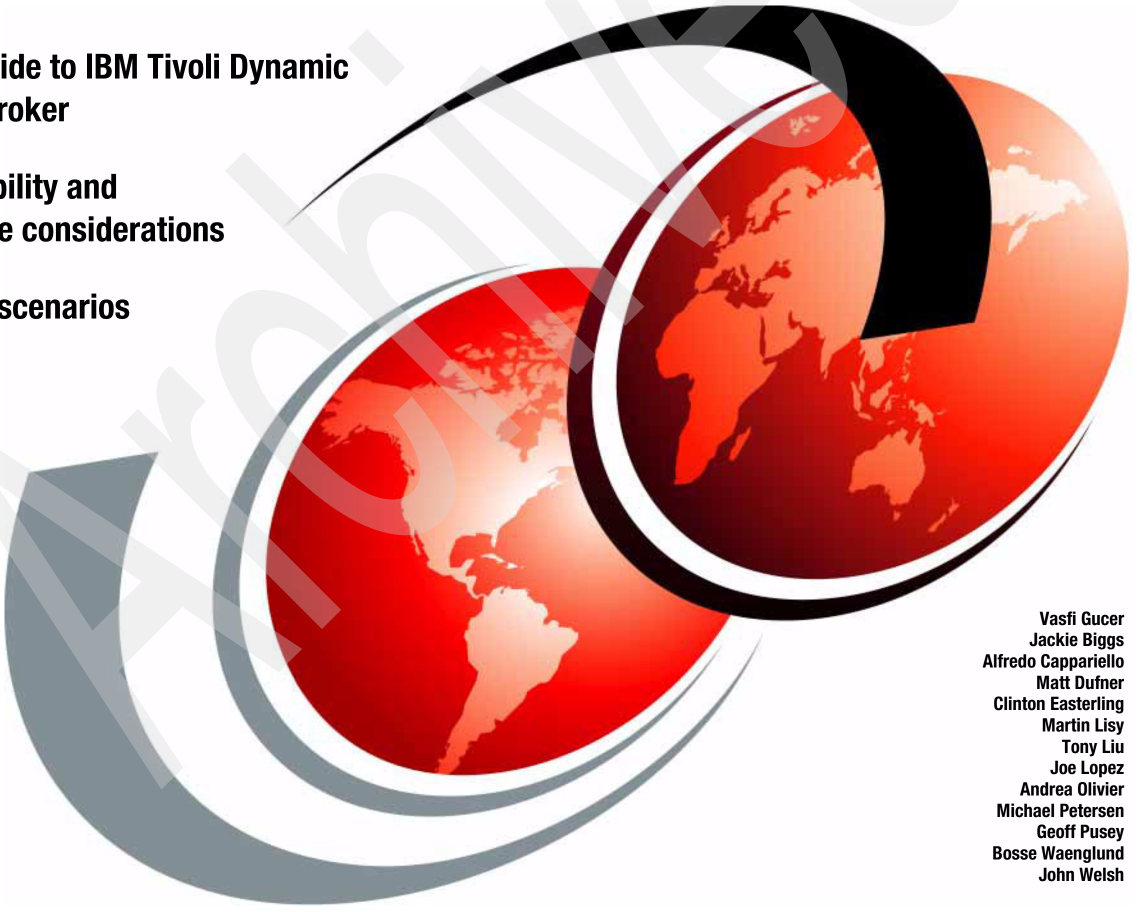
IBM

Getting Started with Tivoli Dynamic Workload Broker Version 1.1

Insider's guide to IBM Tivoli Dynamic Workload Broker

High availability and performance considerations

Integration scenarios



Vasfi Gucer
Jackie Biggs
Alfredo Cappariello
Matt Dufner
Clinton Easterling
Martin Lisy
Tony Liu
Joe Lopez
Andrea Olivier
Michael Petersen
Geoff Pusey
Bosse Waenglund
John Welsh

Redbooks

ibm.com/redbooks

International Technical Support Organization

Getting Started with Tivoli Dynamic Workload Broker Version 1.1

July 2007

SG24-7442-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xiii.

First Edition (July 2007)

This edition applies to IBM Tivoli Dynamic Workload Broker Version 1.1.

Contents

Notices

Trademarks

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

Redbooks (logo) ®
iSeries®
pSeries®
z/OS®
z/VM®
zSeries®
AIX 5L™
AIX®
Cloudscape™
Collation®
CICS®
Domino®
DB2 Universal Database™
DB2®
Enterprise Workload Manager™
Geographically Dispersed
Parallel Sysplex™
HACMP™
IBM®
LoadLeveler®
Lotus®
MVS™
NetView®
OS/400®
Parallel Sysplex®
Rational®
Redbooks®
S/390®
Tivoli Enterprise™
Tivoli Enterprise Console®
Tivoli Management
Environment®
Tivoli®
TME®
WebSphere®

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

NOW, and the Network Appliance logo are trademarks or registered trademarks of Network Appliance, Inc. in the U.S. and other countries.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

EJB, Java, JDBC, JDK, JNI, JRE, J2EE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows Server, 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.

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

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

Preface

IBM® Tivoli® Dynamic Workload Broker (Tivoli Dynamic Workload Broker) is a key element in a comprehensive, on demand, Tivoli workload automation portfolio. It can use dynamic resource information as well as recommendations from other products to determine the best systems to which new jobs will be dispatched.

This IBM Redbooks® publication documents the architecture, installation and customization, operation best practices, performance optimization, high availability considerations, Web Services interface, and troubleshooting of Tivoli Dynamic Workload Broker V1.1.

In addition, we cover integration scenarios with other IBM products, such as IBM Tivoli Workload Scheduler, IBM Tivoli Provisioning Manager, IBM Tivoli Change and Configuration Management Database, IBM Tivoli Monitoring, Tivoli Enterprise™ Portal, and IBM Enterprise Workload Manager™.

Finally, we discuss Tivoli Dynamic Workload Broker operation in a IBM Tivoli Workload Scheduler for a z/OS® end-to-end environment.

Clients and Tivoli professionals who are responsible for installing, administering, maintaining, or using IBM Tivoli Dynamic Workload Broker will find this book a major reference.

The team that wrote this IBM Redbook

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

Vasfi Gucer is a Project Leader at the ITSO, Austin Center. He worked for IBM Turkey for 10 years and has been with the ITSO Austin Center since January 1999. He has more than 12 years of experience in the areas of systems management, networking hardware, and software on mainframe and distributed platforms. He has worked on various Tivoli customer projects as a Systems Architect and Technical Project Manager. He writes extensively and teaches IBM classes worldwide on Tivoli software. He is also an IBM Certified Senior IT Specialist.

Jackie Biggs-Finstad has worked with Tivoli Workload Scheduler on UNIX® and Windows® since 1998. She has held various positions while working with

Tivoli Workload Scheduler Consultant, Systems Engineer, and Product Evangelist. Jackie is currently a Customer Solutions Engineer on the Software Advanced Technologies (SWAT) team, where she provides on-site technical support to the technical sales teams on Proof of Concepts for IBM Tivoli Workload Automation solutions.

Alfredo Cappariello is a member of the development team for the Tivoli Workload Scheduler products in the IBM SWG Rome Lab, Italy. He joined the team in 2004 after some years of experience as a Software Engineer in the Tivoli monitoring area. Alfredo's area of expertise includes installation and deployment. Before joining IBM he worked in the nuclear engineer field as Researcher.

Matt Dufner is a member of Level 2 Support for Tivoli Workload Scheduler in Austin, Texas. Before working in technical support, Matt worked for six years on various Tivoli projects in a software verification role. Before working in verification, he worked as a contractor for IBM, supporting the IBM AIX® operating system for two years.

Clinton Easterling has been working with Tivoli Workload Scheduler for the past six years. Currently, he is the Technical Lead Engineer for the L2 Support team in Austin, Texas. Clint is a Tivoli Workload Scheduler Certified Deployment Specialist and he is also ITIL® certified. Before working for Tivoli, Clint worked for IBM AIX L2 Support and served eight years in the U.S. Military. Clint's areas of expertise include installation and tuning, non-defect problem determination, and Tivoli Workload Scheduler for Applications PeopleSoft® and SAP®. Clint is also a member of the Tivoli Workload Scheduler Users Conference (ASAP) group and technical presenter at conferences such as Tivoli Technical Users Conference and ASAP.

Martin Lisy has been working for IBM since 1995. He is a Systems Management Specialist in the area of the Tivoli Enterprise Management Solutions. Martin has worked on various Tivoli customer projects as the Solution Designer and Implementor. His area of expertise is Tivoli Workload Scheduler, for which he has created various scripting functions that enhance the product capabilities. Martin is an IBM Certified Deployment Professional for Tivoli Workload Scheduler V8.2 and V8.3 and an IBM Certified Deployment Professional for Tivoli Enterprise Console® V3.9. He works with IBM enablement teams in the area of Workload Management, where he is focusing on requirements definitions for new releases of Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker. He also participates in Tivoli certification tests development. Martin holds an engineering degree in Computer Science from VSB-Technical University of Ostrava.

Tony Liu is an IBM Certified Consulting IT Specialist with a speciality in the IBM workload automaton portfilio of products, He has a 29-year career with IBM. He has worked with clients and customers in the Western Region of North America. He is an IBM Certified Deployment Professional - Tivoli Workload Scheduler, is

ITIL Foundation Certified, and has a Linux® Professional Institute LPIC-1 Certification.

Joe Lopez has worked as an IBM Tivoli Workload Scheduler L2 Staff Software Engineer since 2002. He is certified in Tivoli Workload Scheduler V8.2, V8.3, and ITIL. Before this position, he worked as an IBM - Certified Lotus® Professional and Tivoli Business Partner Account Manager for three years.

Andrea Olivier is currently on the Level 2 Support Team for Tivoli Workload Scheduler in Austin, Texas. She has been with Tivoli Support for seven years and is ITIL Foundation Certified. Her previous Tivoli product experiences include Tivoli Framework and IBM Tivoli Composite Application Manager for Response Time Tracking. Prior to Tivoli Support, Andrea worked for IBM AIX Level 2 Support for four years.

Michael Petersen is an Advisory Software Engineer supporting IBM Tivoli Workload Scheduler. Mike has experience in software application development and test for products as well as internal applications. He has assisted customers as an international consultant for IBM library management products. He participated in the IBM Faculty Loan Program. Mike has a BS degree from Purdue University in Mathematics with a double major in Computer Science, an MS degree from Purdue University in Computer Science, and an MBA from Marist College.

Geoff Pusey is a Senior IT Specialist and has been with Tivoli since January 1998 when Unison Software was acquired by Tivoli Systems. Geoff has been working with the Tivoli Workload Scheduling product for the last thirteen years in a consultancy role. This is in the customer training, implementation, and customization of Tivoli Workload Scheduler, which involved creating customized scripts to generate specific reports or to enhance Tivoli Workload Scheduler with a function that is not in the current product.

Bosse Waenglund is a Senior IT Specialist in IBM Global Technology Services in Copenhagen, Denmark. He has 17 years of experience working with IBM Tivoli Workload Scheduler for z/OS. Bosse does consultation and services at customer sites, as well as IBM Tivoli Workload Scheduler for z/OS training. He has worked at IBM for 19 years. His areas of expertise include IBM Tivoli Workload Scheduler for z/OS, IBM Tivoli Dynamic Workload Broker, Geographically Dispersed Parallel Sysplex™, z/OS Workload Manager, and z/OS.

John Welsh is a certified IT Specialist from Atlanta, Georgia. He works in the IBM TechWorks and has been with Tivoli/IBM since 1997, when Unison Software was acquired by Tivoli Systems. He has worked with the IBM Tivoli Workload Scheduling/Maestro product since 1996. John was also a software developer for over eight years at United Technologies Hamilton Standard.

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

Arzu Gucer
International Technical Support Organization, Austin Center

Octavian Lascu, Julie Czubik
International Technical Support Organization, Poughkeepsie Center

Finn Bastrup Knudsen
IBM Denmark

Angelina Ascone, Fabio Benedetti, Umberto Caselli, Paolo Deidda, Xavier Giannakopoulos, Daniela Giri, Antonella Godino, Giulio Santoli, Marco Sebastiani, Tullio Tancredi, Paola Di Vitantonio
IBM Italy

Alan Bivens, Mark Fantacone, Bob Hitson, David Oswald
IBM USA

The team would like to express special thanks to Creighton Hicks, Warren Gill, and Edson Manoel from the IBM USA Tivoli SWAT Team for their contribution in customizing the Tivoli Sytem Automation and Tivoli Change and Configuration Management environments.

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 will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our IBM 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 Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

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

1

Tivoli Dynamic Workload Broker overview

Tivoli Dynamic Workload Broker is a key element in a comprehensive, on demand, Tivoli Workload Automation family portfolio. In this chapter we cover the following topics:

- “Market trends and directions” on page 2
- “Business solutions” on page 3
- “Major functions of Tivoli Dynamic Workload Broker” on page 5
- “Business scenarios” on page 7
- “Technical overview” on page 16

1.1 Market trends and directions

The number of organizations replacing manual, time-consuming workload tasks with automated capabilities continues to increase at a rapid pace. This is done with good reason. When performed manually, these tasks can quickly overwhelm IT staff, increase overall costs, and have a negative effect on service level agreements (SLA).

The adoption of Grid computing for business applications is expanding in various sectors such as insurance, manufacturing, and financial industries. The Grid computing environment is a complex environment for business solutions that require workload automation, job scheduling, and processing. It is complex to address the need for a central point of control to prioritize, manage, and integrate workloads with data for enterprise jobs. In addition to this complexity is the requirement to match computing resources to workload requests in a dynamic fashion without human intervention. Accuracy, efficiency, and flexibility are difficult to maintain.

The market is evolving into a virtual computing environment where clustering, scheduling, and managing workload automation are needed within this virtual environment. Service execution becomes important for delivering operational services to the IT infrastructure and the enterprise. Traditional management tools such as a job scheduler, load balancer. and cluster manager can automate some of these activities. However, these tools leave IT organizations without the ability to automate, dynamically manage, or optimize service execution activities end-to-end, across the enterprise.

While the service execution process is straightforward, it is exceptionally difficult to manage. Management tools must be able to support business processes, enable planned changes to business processes and the IT infrastructure, adapt to unplanned changes in the IT infrastructure, maximize workload velocity, optimize the utilization of IT resources, adhere to stringent SLAs, and ensure compliance. Additional difficulties arise from trying to manage complex, heterogeneous applications and systems as well as processes across organizational silos. Finally, IT organizations that are already limited in resources must somehow deal with a growing number of mixed, interdependent, and often unpredictable workloads that need to be scheduled in real-time, near-real-time, or batch modes.

Enterprises are embracing the service-oriented architecture (SOA) common programming methodology to provide business applications across heterogeneous platforms, operating systems, and data sources. There is the need to provide a batch-on-grid solution to improve business efficiency, utilize resources, reduce costs, and achieve service level agreements for job execution.

Workload automation solutions should be supported by SOA, an open framework that enables IT organizations to easily build, deploy, and integrate business and IT workload processes. SOA can be key to dynamically reconfigure the delivery of services according to business demand; helping to foster innovation and better align IT to business goals. SOA also enables open interfacing for the simplified integration of workload automation into the application and systems management paradigms.

As the market evolves, the requirements for a job scheduling solution evolve too. The traditional scheduling that uses a calendar-based production workload planning and batch job execution is changing. Business needs for new scheduling requirements for platform or application agnostic scheduling, a single consolidated view of all scheduling points, event-based scheduling, and the use of virtualization with Grid computing technology exists today. These requirements will continue to evolve into a service-based scheduling. These requirements follow a process-driven model within a dynamic topology that uses service-oriented architecture integration or adapters based on standards such as Web services.

These challenges can be effectively addressed by a dynamic end-to-end workload automation solution that provides a virtual point of control to implement standardized service execution processes and support a SOA. Solutions should be designed to respond quickly to service demands and changes in the IT environment, balance computing costs and service levels, and improve utilization of IT capacity. The right management tools should be able to:

- Support business processes and policies.
- Enable planned changes to business processes and the IT infrastructure.
- Adapt to unplanned incidences in the IT infrastructure.
- Maximize workload velocity.
- Optimize the utilization of IT resources.
- Adhere to stringent service level agreements (SLAs).
- Ensure adherence to compliance and governance requirements.

1.2 Business solutions

Automating resource-intensive tasks like production workload scheduling will help reduce IT management complexity and lower your overall total cost of ownership (TCO). End-to-end workload automation helps you manage and coordinate up to hundreds of thousands of workloads by executing the correct workload at the correct time and in the correct sequence. By automating workloads, you can significantly increase throughput into existing IT resources while meeting strict service level agreements (SLAs) and easing compliance requirements. So as you automate repeatable processes, you can capture best practices and expertise, and then you can execute processes in a consistent, error-free manner.

The benefits of this solution provide an efficiency to business processes for your enterprise. The results are increasing customer satisfaction, reducing or eliminating complaints from business partners, improving the brand or company image, increasing sales that positively impact the bottom line, and allowing the business to be more competitive or flexible in your industry.

IBM Tivoli workload automation solutions enable you to centrally manage IT workloads with complex dependencies that span multiple applications, systems, and business units, which include mainframe, distributed, Grid, and high-performance computing environments. As a result, you can dynamically route workloads to the best available resources, in real time and in line with changing business demands. IBM Tivoli Dynamic Workload Broker is a product solution for this dynamic workload requirement.

The Tivoli Dynamic Workload Broker eliminates the manually intensive process of workload assignments across multiple, heterogeneous resources. It does this by dynamically routing workloads to the best available resources as defined in a policy. The Tivoli Dynamic Workload Broker helps organizations intelligently manage cross-enterprise workloads and resources by providing a central point to prioritize, manage, and integrate workloads across heterogeneous operating environments. The Tivoli Dynamic Workload Broker increases workload velocity through existing assets, reduces labor-intensive processes by automatically adapting workload execution to changes in the IT environment, and improves the ability to meet stringent service level agreements. This saves time and money for companies with stringent service level agreements on production control staff responsible for this workload.

The Tivoli Dynamic Workload Broker optimizes the use of the IT infrastructure by constantly analyzing your environment to maintain an up-to-date view of the resources available. The Tivoli Dynamic Workload Broker does brokering of workloads on heterogeneous systems to help improve operational efficiencies. The Tivoli Dynamic Workload Broker also analyzes job requirements, evaluates resources based on such requirements, resolves interdependencies, executes jobs, and then monitors these jobs.

Tivoli Dynamic Workload Broker extends Tivoli workload automation capabilities for distributed, mainframe, and end-to-end environments by providing dynamic optimization of workload processing based on performance of the scheduling infrastructure and workload demands. Tivoli Dynamic Workload Broker helps enterprise customers elevate workload automation by transforming static IT infrastructures into dynamic, virtualized environments. It provides dynamic workload brokering to the best available resources based on existing resource loads and available capacity to help improve operational efficiencies. This helps maximize throughput of existing resources.

1.3 Major functions of Tivoli Dynamic Workload Broker

Tivoli Dynamic Workload Broker implements a job scheduling and brokering infrastructure that provides the following major functions:

- Manages the automatic discovery of computers available in the scheduling domain with their attributes.
- Manages the matching of jobs to appropriate resources based on job requirements and resource attributes.
- Manages the job dispatching to target resources, both physical and virtual, that are capable of running the job.
- Optimizes the use of IT resources.
- Manages resource consumption of a job based on the quantities that it is planned to use while running.
- Optionally allocates the required quantity exclusively to the job while it is running.
- Releases the resources as soon as the job terminates for use by other waiting jobs.
- Uses the common agent services infrastructure to provide an agent that simplifies the administration and deployment of the scheduling components on the endpoint systems that are targets for the jobs.
- Provides an easy-to-use Web user interface for managing the scheduling activities.
- Integrates seamlessly with Tivoli Workload Scheduler (8.2.1 and later) and Tivoli Workload Scheduler for z/OS end-to-end (8.2 and later), enabling full control from Tivoli Workload Scheduler of job submission and life cycle.
- Users are able to manage from Tivoli Workload Scheduler the calendar-based triggering and choreography of the flow of jobs that are run by Tivoli Dynamic Workload Broker.
- Integrates scheduling functions and services in the IBM service-oriented architecture common programming model.

1.3.1 IBM Tivoli workload automation portfolio

Tivoli Dynamic Workload Broker is a key element in a comprehensive, on demand, Tivoli workload automation portfolio. The IBM Tivoli workload automation solutions are designed to fit a wide range of enterprise requirements and needs. The IBM Tivoli workload automation family of products allows IT organizations to establish a virtual control point to build and automate a consistent, predictable, and scalable service execution process across the enterprise. These products consolidate enterprise-wide batch and event-triggered workloads that span multiple applications and systems, helping IT organizations efficiently control and manage cross-enterprise workloads.

The Tivoli workload automation portfolio includes:

- *IBM Tivoli Workload Scheduler and IBM Tivoli Workload Scheduler for z/OS* provide advanced workload planning and choreography services, along with extensive calendar and event-triggering services for real-time, near-real-time and batch workloads. They include open Java™ 2 Enterprise Edition (J2EE™) and Web services interfaces to allow IT organizations to consolidate custom applications and services into the service execution process. They also include IBM Tivoli Dynamic Workload Console, which provides a single, Web-based point of control for the entire workload automation network, including the ability to manage workloads by exception, initiate actions, and generate reports.

  Tivoli Workload Scheduler for z/OS end-to-end is an additional scheduling feature of Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS. With this feature, you can use the two scheduler products together to schedule workloads from your z/OS system and run jobs in both mainframe and distributed environments.
- *IBM Tivoli Workload Scheduler for Applications* extends Tivoli Workload Scheduler software by providing extensive awareness and interfacing to SAP, People Soft, and Oracle® business applications. This allows IT organizations to consolidate ERP application workloads into the service execution process.
- *IBM Tivoli Dynamic Workload Broker* further extends Tivoli Workload Scheduler by matching and routing workloads to the best available resources in an on demand manner. Dynamic brokering is based on critical workload and critical path analysis, load requirements, IT resource availability, and business policies.
- *IBM Tivoli Workload Scheduler LoadLeveler®* delivers the ideal solution for high-performance computing environments such as financial modeling, research, or biotechnology simulations. It can be integrated with Tivoli Workload Scheduler networks through Tivoli Workload Scheduler for Virtualized Data Centers to allow enterprises to consolidate high-performance computing workloads with the service execution process.

- *IBM Tivoli Workload Scheduler for Virtualized Data Centers* provide integration between IBM Tivoli Workload Scheduler and the Open Grid Services Architecture.

1.3.2 Tivoli workload automation integration with IBM products

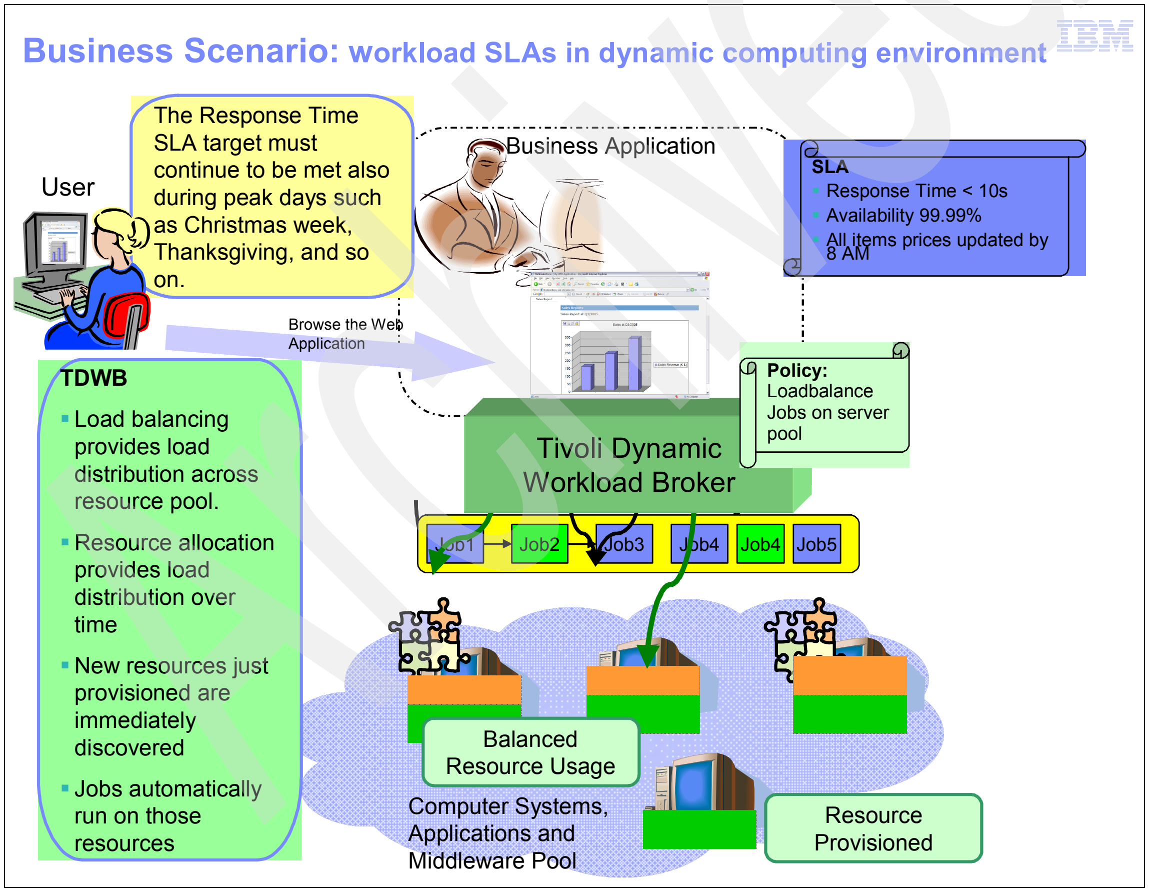
For additional capabilities, the Tivoli workload automation portfolio can be easily integrated with other IBM products:

- IBM Tivoli Enterprise Portal, a Web-based operations console, provides a high-level overview of the entire IT environment.
- IBM Tivoli System Automation for z/OS proactively manages automation, high availability, and performance for z/OS environments.
- IBM Tivoli System Automation for Multiplatforms provides high availability for business applications running in open environments.
- IBM Tivoli Business Systems Manager provides executive dashboards with knowledge of key business processes and real-time service level status from a single console.
- IBM Tivoli Service Level Advisor delivers automated support for SLAs.
- IBM Tivoli Provisioning Manager efficiently provisions and configures servers, operating systems, middleware, applications, and network devices.
- IBM Tivoli Storage Manager automates data backup and restore functions, supporting a broad range of platforms and storage devices, and centralizing storage management operations.
- IBM Workload Manager for z/OS, a component of the IBM z/OS operating system, can identify work requests based on service class definitions, sets performance goals for service classes, and assigns specific IT resource usage constraints to service classes.
- IBM Enterprise Workload Manager, a management product for distributed, open systems, identifys work requests based on service class definitions, tracks performance of work requests, and shifts computing resources as needed to achieve specified performance goals.
- IBM Service Management is a portfolio of products, services, and solutions that automate and manage critical IT processes and enable IT governance.

1.4 Business scenarios

Let us look at four business scenarios with requirements that describe solutions using IBM workload automation portfolio. These scenarios consider the importance of service level agreements during peak periods as a key company goal. This SLA is the contract between the user and the service provider of the business application that articulates the objective for response time, resource availability, and timeframe expected. The service provider uses Tivoli Dynamic Workload Broker as a technical tool to meet the business objectives defined in the SLA.

The Tivoli Dynamic Workload Broker product provides load balancing across a resource pool, resource allocation provides load distribution over time, new resources are discovered, and jobs are automatically run on these resources. Figure 1-1 is an example of a business scenario in a dynamic computing environment and how Tivoli Dynamic Workload Broker addresses the company's SLA. The Tivoli LoadLeveler product provides load balancing capabilities. A scenario is provided to position this product to Tivoli Dynamic Workload Broker.



Business Scenario: workload SLAs in dynamic computing environment
IBM
User
The Response Time SLA target must continue to be met also during peak days such as Christmas week, Thanksgiving, and so on.
Business Application
SLA
Response Time < 10s
Availability 99.99%
All items prices updated by 8 AM
Browse the Web Application
TDWB
Load balancing provides load distribution across resource pool.
Resource allocation provides load distribution over time
New resources just provisioned are immediately discovered
Jobs automatically run on those resources
Policy: Loadbalance Jobs on server pool
Tivoli Dynamic Workload Broker
Job1
Job2
Job3
Job4
Job4
Job5
Balanced Resource Usage
Computer Systems, Applications and Middleware Pool
Resource Provisioned

Figure 1-1 Business scenario: workload SLAs in dynamic computing environment

The four scenarios are:

- Tivoli Workload Scheduler with Tivoli Dynamic Workload Broker
- Tivoli Workload Scheduler with Tivoli Workload Scheduler LoadLeveler
- Tivoli Workload Scheduler for z/OS end-to-end with Tivoli Dynamic Workload Broker
- Tivoli Dynamic Workload Broker as a Web Services solution

1.4.1 Tivoli Workload Scheduler with Tivoli Dynamic Workload Broker

This scenario, which is shown in Figure 1-2, entails a small or medium size business in the health industry. The business challenge is scheduling a series of dependent FTP programs in a strict order. Other programs are dependent upon one another where one or more must finish before the next series of programs run. These programs are scheduled to run based on a certain time or day depending on a work calendar schedule. The programs execute on a cluster of computers arranged in a Grid computing environment of integrated and shared data. There is a requirement to determine which computer in the Grid cluster has the lowest utilization to run the scheduled program.

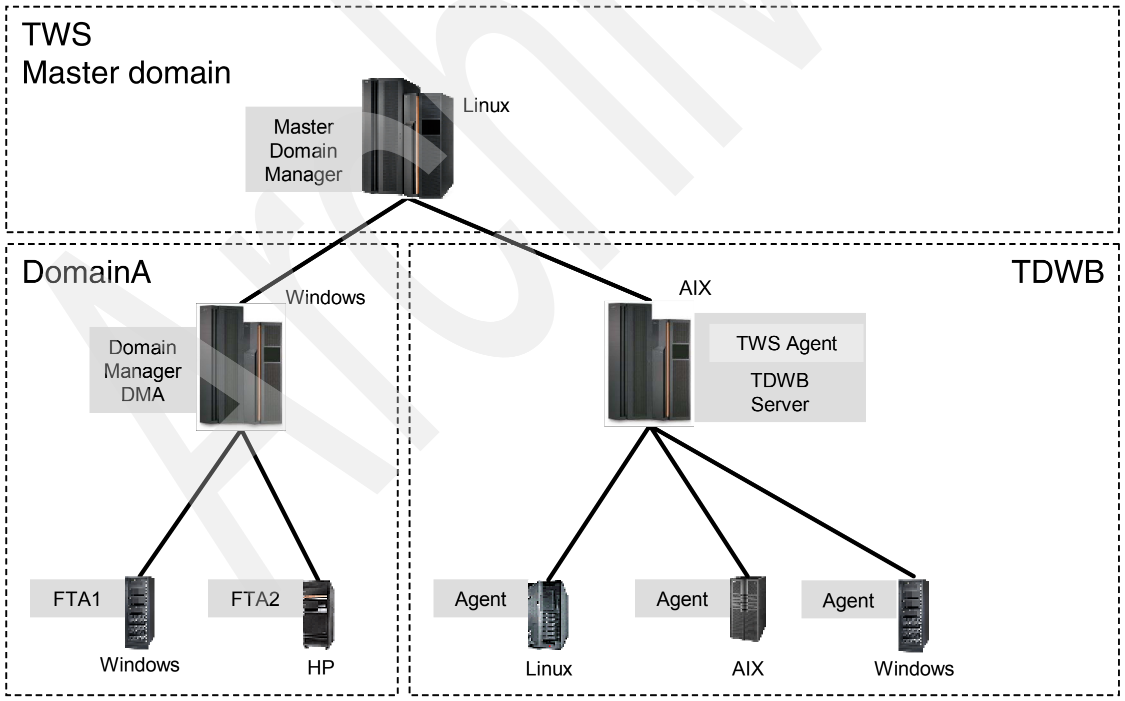


Figure 1-2 Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker architecture diagram

The solution would be for Tivoli Workload Scheduler to provide the planning, executing, and monitoring of these programs. Dependent programs are controlled by Tivoli Workload Scheduler so predecessor and successor programs execute in the appropriate order. The programs are balanced between computers using the Tivoli Dynamic Workload Broker during the execution phase. Tivoli Dynamic Workload Broker is the middleware that determines which computer has the available resources at the point in time needed to run the program. When the program is done, the next successor program is run in the series of predecessor/successor dependent jobs. It will then route the program scheduled as a Tivoli Workload Scheduler job to the appropriate system with the most available resources. This allows better resource utilization without having a negative effect on the service level objectives.

The combined strengths of Tivoli Workload Scheduler for scheduling and Tivoli Dynamic Workload Broker for load balancing and executing jobs allow the software to handle this business challenge. The need for manual human intervention to execute this process is eliminated due to the combination of Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker.

1.4.2 Tivoli Workload Scheduler and Tivoli Workload Scheduler LoadLeveler

The scenario shown in Figure 1-3 on page 11 entails an organization in the airline manufacturing industry. There is a need to schedule independent jobs based on certain calendar days. Sometimes these jobs need to run in parallel, while other times jobs are dependent to finish first before starting the next job or series of jobs. These jobs have different priorities for resources that are available. There is a need to define programs to run in classes that define the characteristics associated with the type of job, such as its priority, run limits, and resources. The management of this organization wants a central console that has a graphical interface for the operators. These operators want an easy-to-use interface that they can point and click to use. Security is a concern where operators are assigned a role-based authorization to use this scheduling system.

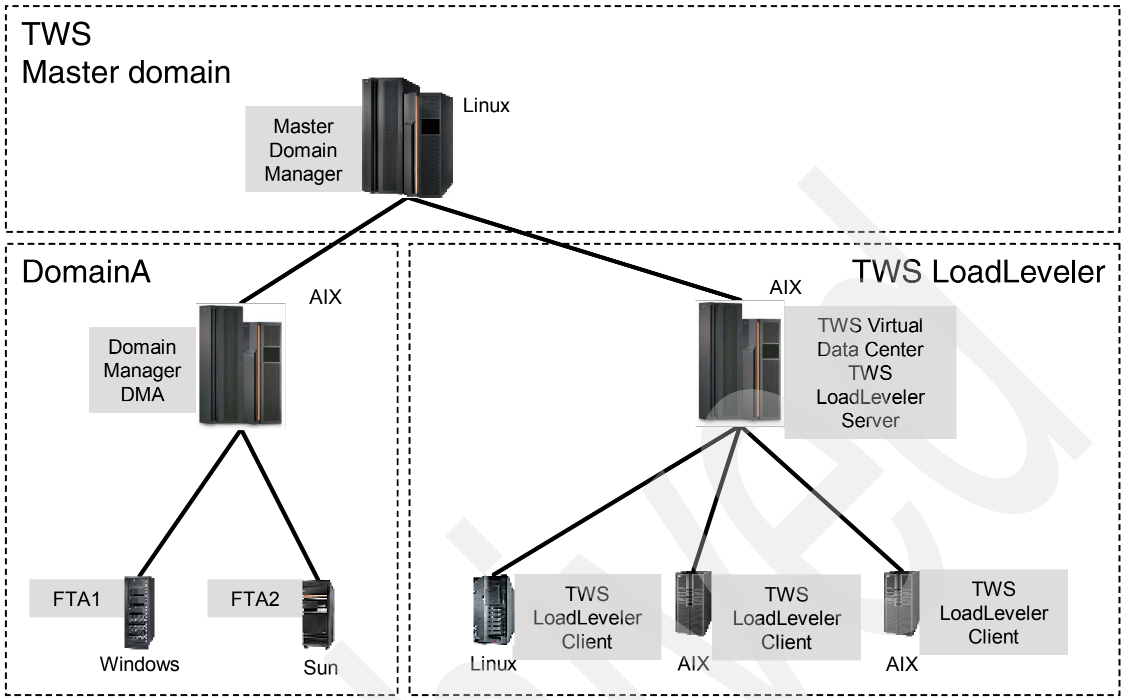


Figure 1-3   Tivoli Workload Scheduler and Tivoli Workload Scheduler LoadLeveler architecture diagram

The solution consists of the Tivoli Workload Scheduler with Tivoli Workload Scheduler LoadLeveler. Tivoli Workload Scheduler has a job scheduling console that provides the ease-of-use interface. The security provided in Tivoli Workload Scheduler will provide the role-based authorization to job scheduling objects such as jobs, calendars, workstations, and so on. The Tivoli Workload Scheduler jobs are submitted to LoadLeveler, thought they are not necessarily executed in the order of submission.

LoadLeveler dispatches jobs based on their priority, resource requirements, and special instructions. For example, administrators can specify that long-running jobs run only on off-hours and that short-running jobs be scheduled around long-running jobs or that certain users or groups get priority. In addition, the resources themselves can be tightly controlled. Use of individual machines can be limited to specific times, users, or job classes or LoadLeveler can use machines only when the keyboard and mouse are inactive.

The combined strengths of Tivoli Workload Scheduler for ease-of-use and role-based security with the Tivoli Workload Scheduler LoadLeveler for load balancing by priority and resources allow the software to handle these requirements. The combined solution provides the planning, executing, and monitoring for running jobs that require scientific calculations.

1.4.3 Tivoli Workload Scheduler for z/OS end-to-end with Tivoli Dynamic Workload Broker

This scenario shown in Figure 1-4 consists of a large corporation in the financial services, communications, and health care industries. The management, operations, and support staff are organized in a centralized job-scheduling model so that all enterprise jobs are handled from one central organization. This reduces system and operational complexity, which leverages IT staff skills and knowledge. This model provides a cost-effective solution regarding the total cost of ownership for scheduling workload handles by a centralized staff.

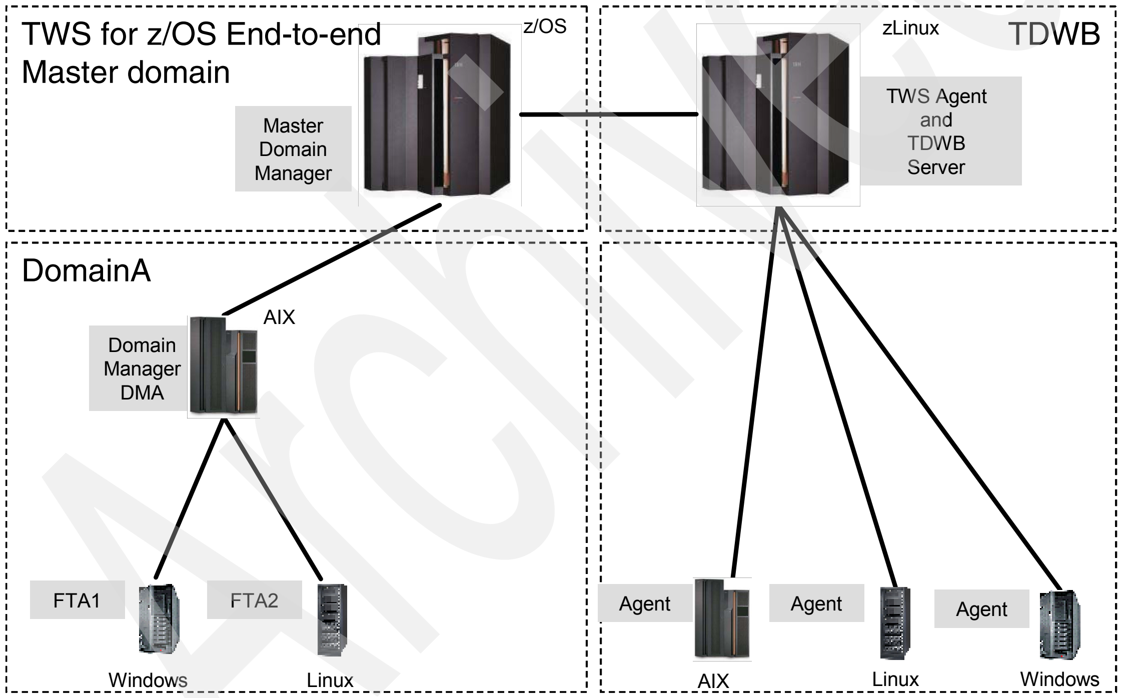


Figure 1-4 Tivoli Workload Scheduler for z/OS end-to-end with Tivoli Dynamic Workload Broker

Senior management has determined that the mainframe is their central platform of choice to reduce the cost of power and cooling of computer systems. This is part of their server consolidation initiative. They decided that the proliferation of distributed servers is not cost effective to stay competitive in their industry. However, they recognize the need for distributed platforms for strategic and necessary business processes. They determined that the mainframe will be the central server for workload automation to complement the distributed environment.

The corporate standard is to use the mainframe as the central platform for the infrastructure of middleware for business process applications. The infrastructure takes into consideration various services such as data solutions services, security and encryption services, services oriented architecture on the mainframe, and the implementation services for Linux-based processes.

For example, the legacy mainframe business applications and subsystems run within the z/OS operating system. Data solution services for customer information uses a relational database, and transaction processing is done on the mainframe. Security and encryption services reside on the mainframe due to the confidentiality of client data for their financial services applications. Business applications incorporate SOA to utilize the reusable processes on the mainframe. Where appropriate, the SOA processes are done within the virtualization model running on z/VM® and zLinux. These mainframe services were incorporated within the workload automation as needed.

Senior management is committed to the virtualization of computing resources that use z/VM and zLinux on the mainframe. The management knows it is more cost effective to run infrastructure software in this virtual environment rather than purchase additional hardware and software for this workload. This initiative is used for the Tivoli workload automation portfolio.

The corporate direction for workload automation is Tivoli Workload Scheduler for z/OS as an end-to-end solution with Tivoli Dynamic Workload Broker. The business challenge is automating the workload balancing and job scheduling within the distributed grid computing environment. This distributed environment has key business applications in a grid environment. These applications use workload load balancing to virtualize workload so that it can be divided and moved around a dynamic IT infrastructure in a variety of UNIX, Linux, and Windows systems. However, there are workloads that need to run in the traditional job scheduling manner that are not dynamic or need load balancing.

The Tivoli Workload Scheduler for z/OS end-to-end solution with the staff reorganization supports a central workload automation initiative and the virtualization of computing resources for these business applications. In Figure 1-4 on page 12 the architecture shows the Tivoli Workload Scheduler for z/OS as the master domain server. It communicates to the AIX domain manager and Fault Tolerant Agents for the traditional job scheduling in DomainA. In the Tivoli Dynamic Workload Broker structure, it shows the zLinux Tivoli Dynamic Workload Broker server that also acts as a Tivoli Workload Scheduler agent. The Tivoli Dynamic Workload Broker server then does the load balancing between the three agents (AIX, Linux, and Windows) under its control.

1.4.4 Tivoli Dynamic Workload Broker as a Web Services solution

This scenario entails a company that sells printed materials from a catalog. This company uses a Web-based customer interface within a supply chain to a business partner that manufactures the customer-ordered product. They receive orders over the Web that starts a process of designing a custom product for their customer and finally sending this order to the factory to manufacture this customer's product. The product is then shipped from the factory to the company to be checked for quality assurance before it is delivered to the customer.

The business challenge is processing these orders dynamically across a pool of heterogeneous hardware and operating systems. These systems are clustered together using shared disk space. These systems consists of Linux and Windows platforms. The problem is routing these orders automatically to a system with resources available to process the custom order prior to sending it to the factory.

The corporate direction is to develop a Web services application for their customers to use. This Web application starts the chain of events that incorporate the business processes into internal systems then to their business partner network. The management is committing application development, infrastructure restructuring, and process re-engineering to provide better customer service level agreements. What management needs is a middleware tool to better utilize the hardware resources in their corporate Grid environment.

The solution incorporates the features of Tivoli Dynamic Workload Broker to dynamically load balance this Web Services application. Once the application developer and administrator/resource developer define the necessary jobs within Tivoli Dynamic Workload Broker, the monitoring is done by the IT operator or scheduling operator. The business benefit is in the execution of this application. Once a customer submits an order via the Web, the jobs execute within this Web Services application balanced between systems controlled by Tivoli Dynamic Workload Broker.

Figure 1-5 depicts the Tivoli Dynamic Workload Broker scheduling life-cycle flow and the tools used by the IT staff in this new order processing and supply chain management application. Let us step through this scheduling life-cycle flow for this Web Services application.

1. The application developer using the Tivoli Dynamic Workload Broker Job Definition Editor defines the jobs that are used in order processing and supply chain management. These jobs are SOA processes such as a database lookup of customer profile data. These jobs are stored in the Tivoli Dynamic Workload Broker Job Repository in step 1.1.

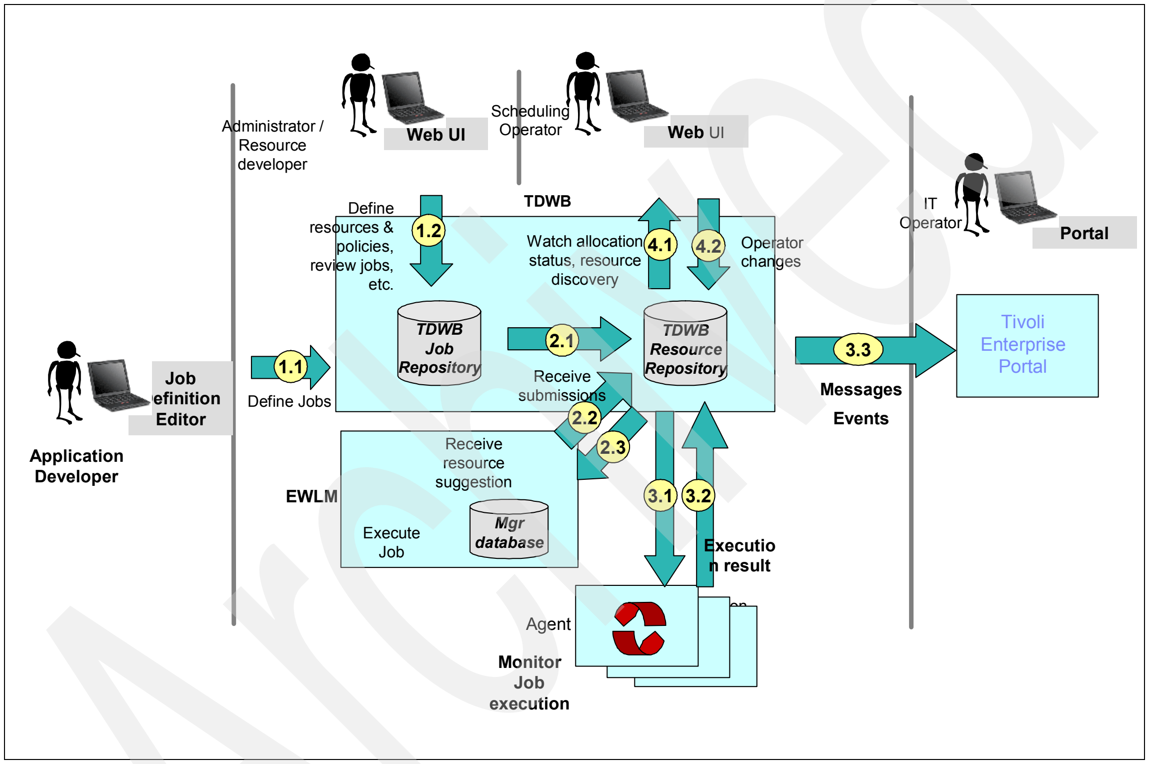


Figure 1-5 Tivoli Dynamic Workload Broker scheduling life-cycle flow

2. The administrator or resource developer will define (using a policy) the resources needed to run these jobs using a Web interface in step 1.2.
3. From the Web application, requests for job submissions are received in step 2.1.
4. The Enterprise Workload Manager (EWLM) is optional and provides additional information about resources, which is suggested to Tivoli Dynamic

Workload Broker Resource Repository to complete the dynamic state of the environment in steps 2.2 and 2.3.

5. Tivoli Dynamic Workload Broker identifies the agent with the best available resources to execute a job in step 3.1. Results of the job execution are submitted back to the Tivoli Dynamic Workload Broker Resource Repository in step 3.2.
6. If there are any exceptions or error messages, these are posted to the Tivoli Enterprise Portal monitored by the IT Operator in step 3.3.

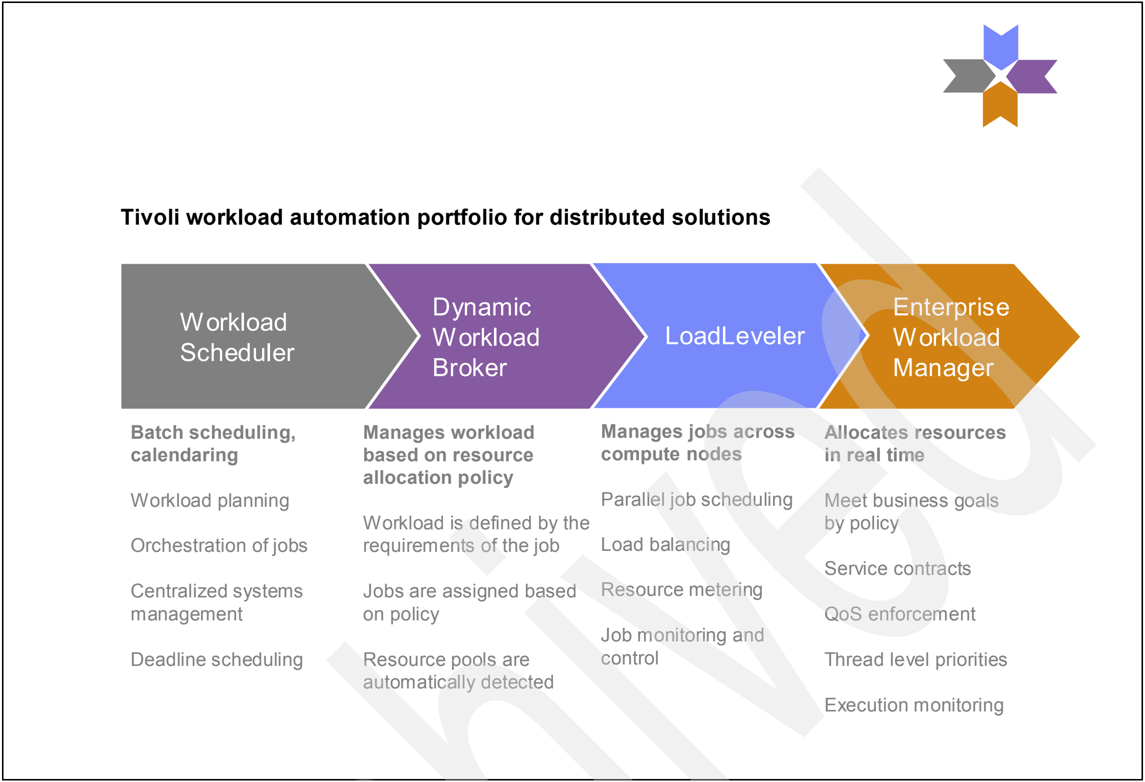
The allocation status or new resource discoveries are posted to the scheduling operator via a Web interface in step 4.1. If any corrective action needs to be done, the scheduling operator can update the Tivoli Dynamic Workload Broker Resource Repository via step 4.2.

1.5 Technical overview

The IBM Tivoli workload automation portfolio provides a solution combining the strengths of various products. This could be thought of as the Tivoli workload management solution set. This technical overview describes various product features and functions for the Tivoli workload automation portfolio for distributed solutions.

The four products for distributed solutions reviewed in this section are Tivoli Workload Scheduler, Tivoli Dynamic Workload Broker, Tivoli Workload Scheduler LoadLeveler, and IBM Enterprise Workload Manager.

Note: IBM Enterprise Workload Manager is not a Tivoli branded product.


Tivoli workload automation portfolio for distributed solutions

Workload Scheduler

Batch scheduling, calendaring

Workload planning

Orchestration of jobs

Centralized systems management

Deadline scheduling

Dynamic Workload Broker

Manages workload based on resource allocation policy

Workload is defined by the requirements of the job

Jobs are assigned based on policy

Resource pools are automatically detected

LoadLeveler

Manages jobs across compute nodes

Parallel job scheduling

Load balancing

Resource metering

Job monitoring and control

Enterprise Workload Manager

Allocates resources in real time

Meet business goals by policy

Service contracts

QoS enforcement

Thread level priorities

Execution monitoring

Figure 1-6 IBM Tivoli workload automation portfolio

Tivoli Workload Scheduler is the centralized job scheduler that allows you to do batch scheduling based on a calendar. You would plan the workload with dependencies between jobs within a job stream over a future time period. This software will then execute, also known as orchestrate, these job streams and allow centalized management for successful or exception handling for unsuccessful job execution. If necessary due to workload delays, deadline scheduling can be handled by the software. Tivoli Workload Scheduler can be specified to suppress, continue, or cancel a job or job stream processing for the desired results.

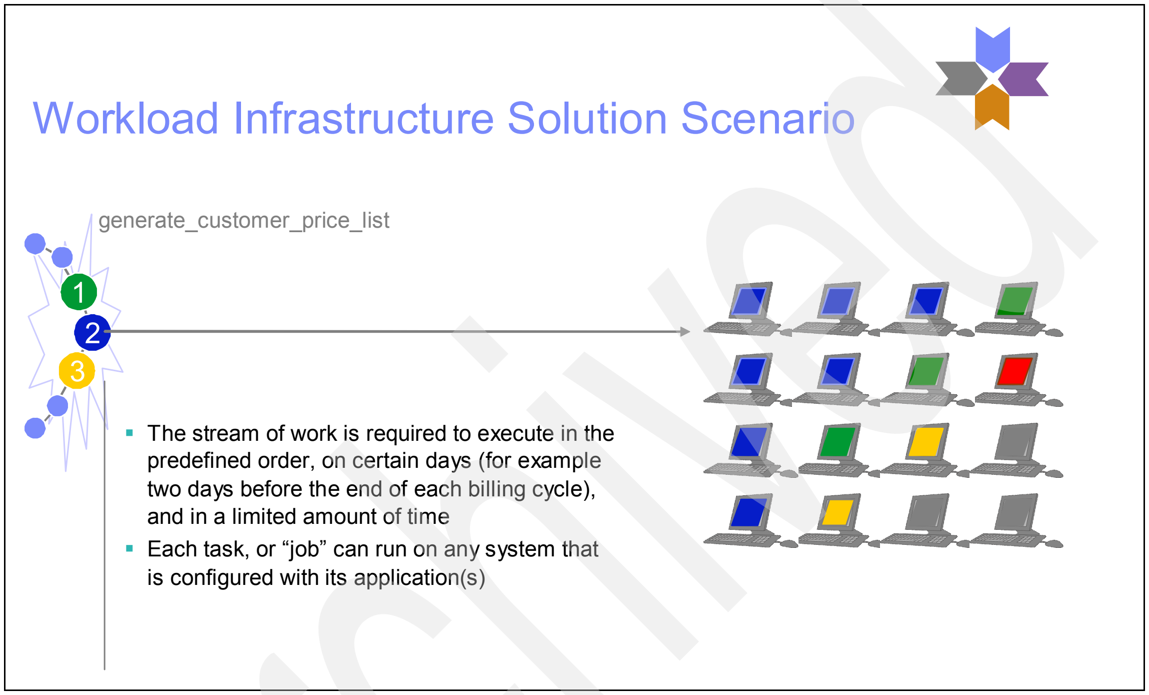
Tivoli Dynamic Workload Broker manages the workload submitted to it based on a resource allocation policy. A policy is defined by the workload administrator by criteria of available resources. The Tivoli Dynamic Workload Broker server software determines what hardware has the available resources required by the job. Clustered systems pooled together have Tivoli Dynamic Workload Broker agents that communicate with the server. These agents automatically detect the current state of resources. The server then determines the system with the agent that has the resources based on the policy requirements for the job. The job is assigned and run on this resource. When completed, the agent communicates back to the server the job status.

Tivoli Workload Scheduler LoadLeveler (or Tivoli LoadLeveler) is a parallel scheduling system that matches each job's processing needs and priority with available resources and special instructions for maximum resource utilization. It can track the total resources used by each serial or parallel job and offers several reporting options to track jobs and utilization by user, group, account, or type over a specified time period. LoadLeveler offers job checkpointing and suspension with optional job cancellation, hold, and re-queue. These capabilities provide great flexibility in defining real-time job and resource priority control. It delivers tight control of resources so that it can limit the use of individual machines to specific times, users, or job classes, or can use machines only when the keyboard and mouse are inactive. Tivoli LoadLeveler also provides a single point of control for effective workload management, offers detailed accounting of system utilization for tracking or chargeback, and supports high availability configurations.

Enterprise Workload Manager is an implementation of policy-based performance management. The scope of management is a set of servers that you logically group into what is called an Enterprise Workload Manager Management Domain. The set of servers included in the Management Domain has some type of relationship, for example, the set of servers supporting a particular line of business. The line of business may consist of multiple business processes spread across a few servers or a thousand servers. There is a management focal point for each Enterprise Workload Manager Management Domain, called the Enterprise Workload Manager domain manager. The domain manager coordinates policy actions across the servers, tracks the state of those servers, and accumulates performance statistics on behalf of the domain.

On each server (operating system) instance in the management domain there is a thin layer of Enterprise Workload Manager logic installed called the Enterprise Workload Manager managed server. From one perspective the managed server layer is positioned between applications and the operating system. The managed server layer understands each of the supported operating systems, gathering resource usage and delay statistics known to the operating system. A second role of the managed server layer is to gather relevant transaction-related statistics from middleware applications. The application middleware implementations, such as WebSphere® Application Server, understand when a piece of work starts and stops, and the middleware understands when a piece of work has been routed to another server for processing, for example, when a Web server routes a servlet request to a WebSphere Application Server.

The managed server layer dynamically constructs a server-level view describing relationships between transaction segments known by the applications with resource consumption data known by the operating system. A summary of this information is periodically sent to the domain manager, where the information is gathered together from all the servers in the Management Domain to form a global view. See Figure 1-7.



Workload Infrastructure Solution Scenario

- The stream of work is required to execute in the predefined order, on certain days (for example two days before the end of each billing cycle), and in a limited amount of time
- Each task, or “job” can run on any system that is configured with its application(s)

Figure 1-7 Workload infrastructure solution scenario

Let us use a workload infrastructure solution scenario to understand how a job is handled differently using the Tivoli workload automation portfolio for distributed systems. The generate_customer_price_list is a stream of work made up of a series of jobs. This stream of work has a predefined order for which the jobs are required to run on a certain day in a limited amount of time. A job will run on a system that is configured with the resources and data to run the application software.

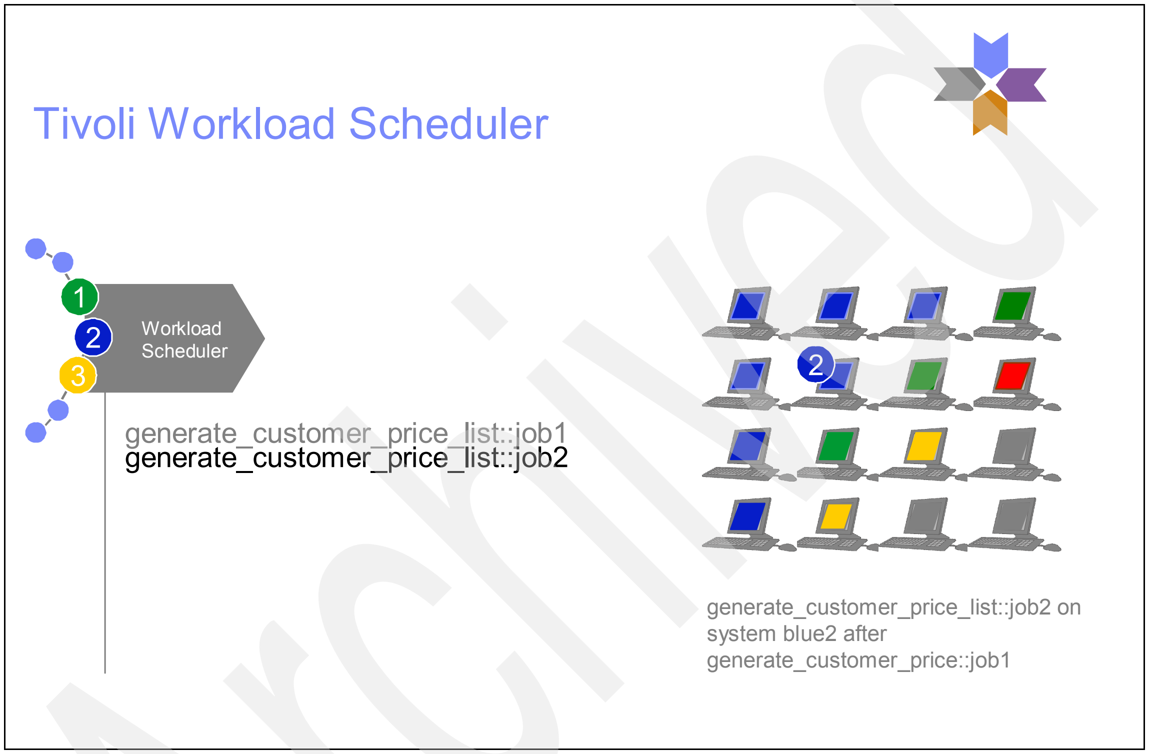

Tivoli Workload Scheduler
1
2
3
Workload Scheduler
generate_customer_price_list::job1
generate_customer_price_list::job2
2
generate_customer_price_list::job2 on system blue2 after generate_customer_price::job1

Figure 1-8 Tivoli Workload Scheduler

Tivoli Workload Scheduler orchestrates work flows with calendaring, end-to-end dependency management, and fault tolerance. Correct dependency resolution is ensured, and production is automated with timelines, deadlines, and control. Each task, or *job* is defined to only one system, and always runs there regardless of dynamic conditions. In this case, generate_customer_price_list::job2 will run only on system blue2, though other *blue* systems could be available or could better handle this job. Tivoli Workload Scheduler schedules generate_customer_price_list:job2 to run. However, load balancing this job onto another blue system is done by another Tivoli workload automation product described below. See Figure 1-9.

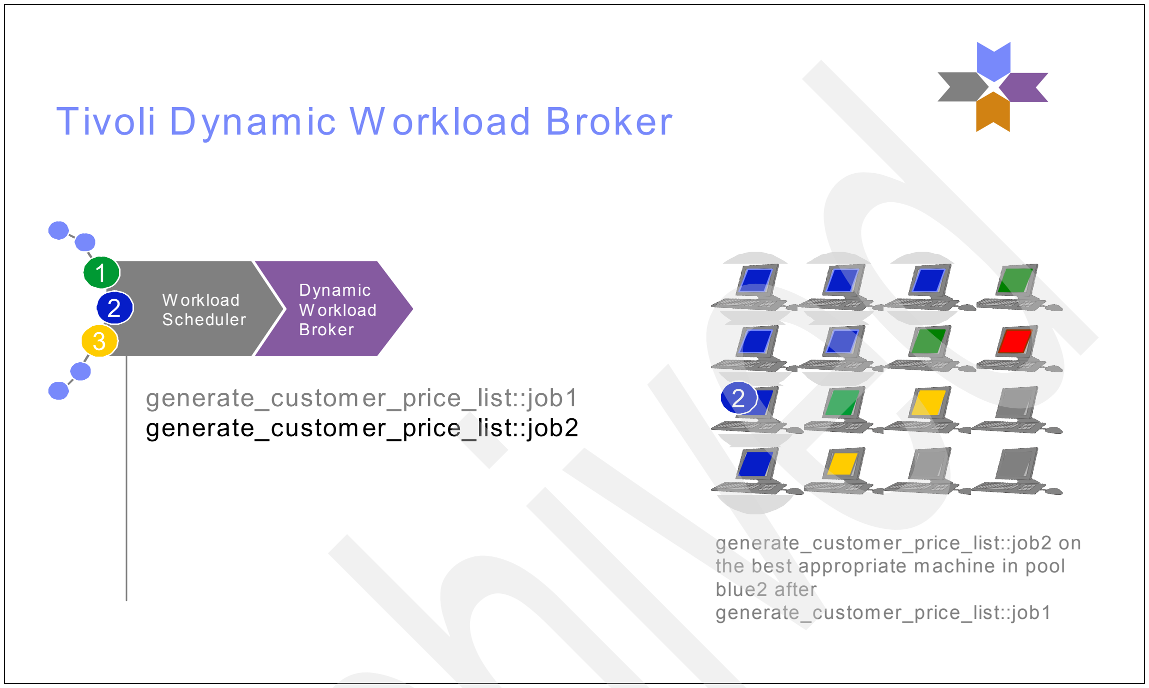

Tivoli Dynamic Workload Broker
1
2
3
Workload Scheduler
Dynamic Workload Broker
generate_customer_price_list::job1
generate_customer_price_list::job2
2
generate_customer_price_list::job2 on the best appropriate machine in pool blue2 after generate_customer_price_list::job1

Figure 1-9 Tivoli Dynamic Workload Broker

Tivoli Dynamic Workload Broker automatically discovers and chooses the best system for executing the job, based on current system parameters such as availability, memory, disk, and CPU utilization. These parameters are defined in a policy used by Tivoli Dynamic Workload Broker. So when the generate_customer_price_list::job2 is scheduled to run, the best available system is selected from a pool of blue2 systems by the Tivoli Dynamic Workload Broker software policy. Therefore, Tivoli Worload Scheduler can schedule this job and Tivoli Dynamic Workload Broker determines at that time the best blue system to run that job. See Figure 1-10.

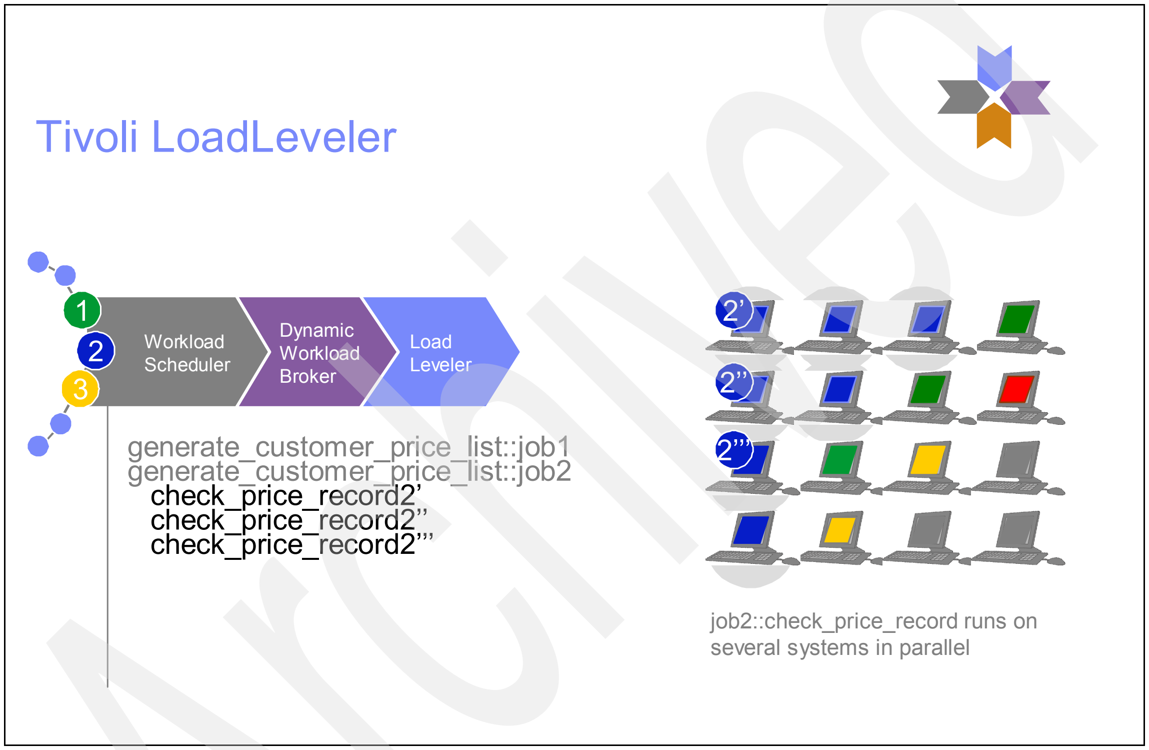


Figure 1-10 Tivoli LoadLeveler

Tivoli LoadLeveler manages the complex message handling among multiple cluster nodes that simultaneously perform computations on massively parallel jobs. For example, generate_customer_price_list::job2 spawns three separate jobs to check a price record. These three check_price_record jobs run in parallel and are load balanced amongst a pool of blue2 systems under the direction of Tivoli LoadLeveler. See Figure 1-11.

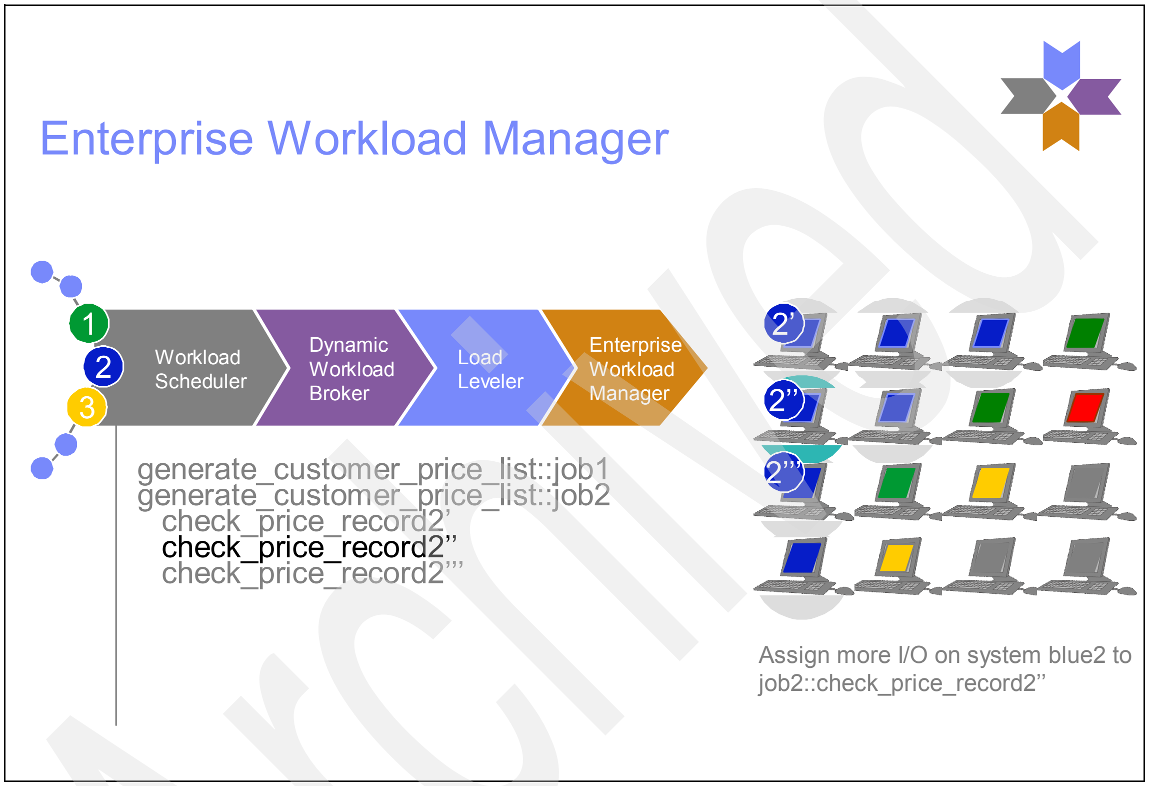

Enterprise Workload Manager
1
2
3
Workload Scheduler
Dynamic Workload Broker
Load Leveler
Enterprise Workload Manager
generate_customer_price_list::job1
generate_customer_price_list::job2
check_price_record2'
check_price_record2''
check_price_record2'''
2'
2''
2'''
Assign more I/O on system blue2 to job2::check_price_record2''

Figure 1-11 Enterprise Workload Manager

Enterprise Workload Manager ensures that the systems with critical workload have the correct amount of system resources for high-priority applications. Predictive advice for compute node processing is done so that resources are made available to applications to meet service-level objectives. An important aspect of Enterprise Workload Manager is that all data collection and aggregation activities are driven by a common service level policy, called the Enterprise Workload Manager Domain Policy. This policy is built by an administrator to describe the various business processes that the domain supports and the performance objectives for each process. In this example, the domain policy determines that system blue2 running the check_price_record2 job needs more I/O, so the Enterprise Workload Manager then assigns additional I/O resource.

2

Tivoli Dynamic Workload Broker architecture

While the concepts of Tivoli Dynamic Workload Broker are easily understandable, from the architectonical perspective Tivoli Dynamic Workload Broker can be a complex application.

In this chapter we provide detailed descriptions of Tivoli Dynamic Workload Broker components and interactions among them. We also describe the user interfaces used for managing the workload across the IT environment.

We focus especially on the following topics:

- "Topological view" on page 27
- "Major server components" on page 28
- "Tivoli Dynamic Workload Broker agent" on page 34
- "Common Agent Services" on page 38
- "Job and resource definitions" on page 43
- "Tivoli Dynamic Workload Broker User interfaces" on page 44
- "Security features" on page 51
- "Tivoli Dynamic Workload Broker deployment scenarios" on page 66

- “Physical location of Tivoli Dynamic Workload Broker’s components” on page 73

While describing the Tivoli Dynamic Workload Broker components and the other related software, we discuss default and optional installation locations. Because this chapter describes the Tivoli Dynamic Workload Broker architecture, we do not list the software and hardware prerequisites. You need to be aware that Tivoli Dynamic Workload Broker components can only be installed only on systems that meet both hardware and software prerequisites of the product.

Furthermore, we describe the following deployment scenarios:

- Tivoli Dynamic Workload Broker standalone solution

  In 2.8, “Tivoli Dynamic Workload Broker deployment scenarios” on page 66, we discuss the possible installation locations of Tivoli Dynamic Workload Broker components.
- Common usage of Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler (TWS) on distributed platforms
- Tivoli Dynamic Workload Broker usage with enterprise monitoring

These topics are discussed from anarchitectonical perspective. Deeper technical scenarios are included in Chapter 8, “Integration with other IBM Tivoli products” on page 301. A deployment scenario of Tivoli Dynamic Workload Broker integration with Tivoli Workload Scheduler end-to-end is described in Chapter 11, “Managing Tivoli Dynamic Workload Broker jobs using Tivoli Workload Scheduler for z/OS end-to-end” on page 515.

2.1 Topological view

The central point of Tivoli Dynamic Workload Broker topology is a managing *server*, which manages its *agents*. The users interact with the server through user interfaces (also called *clients*).

The main purpose of Tivoli Dynamic Workload Broker server is to determine the best available resource for running each submitted job. All of the logic necessary for accomplishing this task is centralized on the *server*. After determining the best available resource, a job is submitted to the target system utilizing the Tivoli Dynamic Workload Broker *agent*. Each Tivoli Dynamic Workload Broker agent provides several services: launches/cancels jobs, returns job outputs, returns information about hosting system utilization (such as CPU load and memory usage), and so on.

Figure 2-1 shows the Tivoli Dynamic Workload Broker's topology.

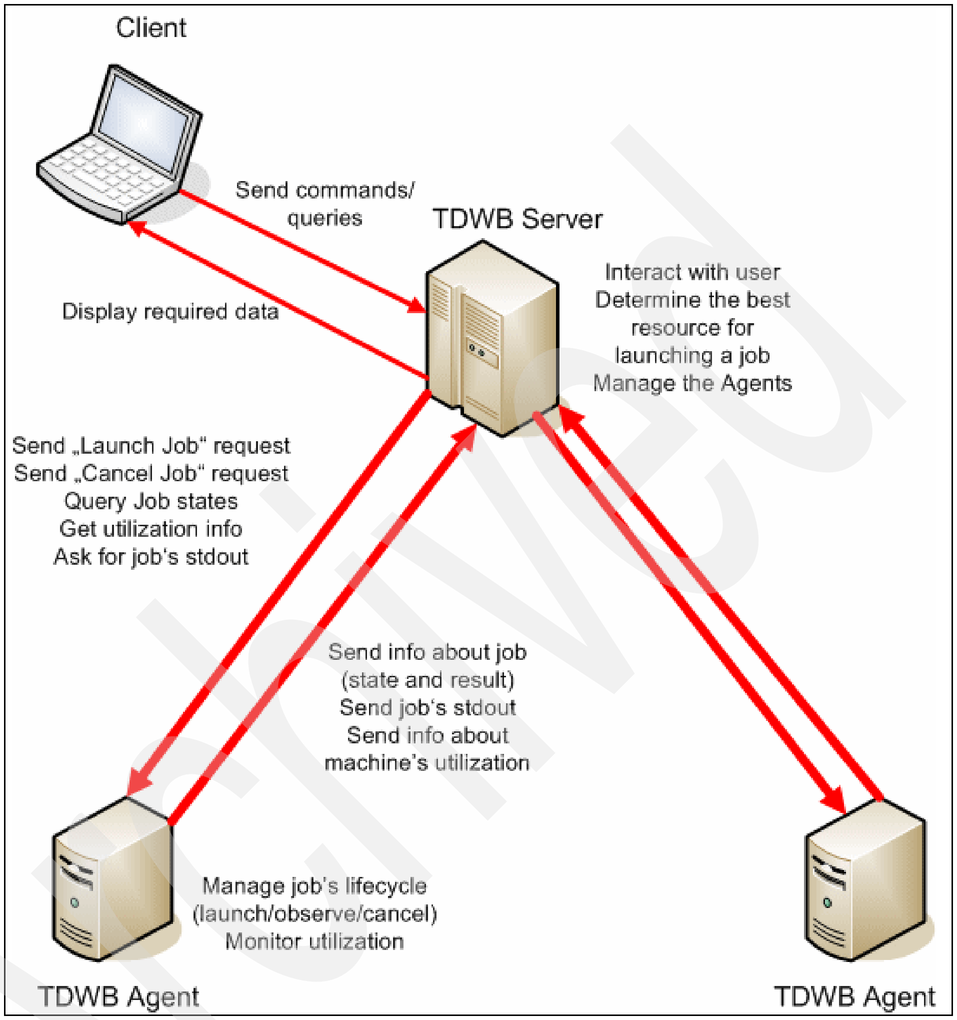

Client
Send commands/ queries
TDWB Server
Display required data
Interact with user
Determine the best resource for launching a job
Manage the Agents
Send „Launch Job" request
Send „Cancel Job" request
Query Job states
Get utilization info
Ask for job's stdout
Send info about job (state and result)
Send job's stdout
Send info about machine's utilization
Manage job's lifecycle (launch/observe/cancel)
Monitor utilization
TDWB Agent
TDWB Agent

Figure 2-1 Tivoli Dynamic Workload Broker topology

2.2 Major server components

In this section we describe the server components of Tivoli Dynamic Workload Broker.

The Tivoli Dynamic Workload Broker server is an J2EE enterprise application installed into the IBM WebSphere Application Server. It consists of several

WebSphere enterprise applications and uses a RDBMS as its persistent data storage.

The core WebSphere enterprise application is named *ITDWB*, and from a logical point of view is divided into the following parts:

- Resource Repository
- Resource Advisor
- Job Dispatcher
- Job Repository
- Allocation Repository

Tivoli Dynamic Workload Broker leverages the *Common Agent Services (CAS)* as its server-agent communications infrastructure. Tivoli Dynamic Workload Broker server integrates with *Agent Manager*, which is the server side of Common Agent Services. The Agent Manager is a mandatory component, but the installation location will vary depending on your architecture. Agent Manager is an enterprise application running on the WebSphere Application Server. Depending on the deployment scenario it can be installed either in the same WebSphere Application Server as the Tivoli Dynamic Workload Broker server or on another WebSphere Application Server instance that satisfies the software prerequisites. The more detailed description of Common Agent Services is included in 2.4, "Common Agent Services" on page 38.

Additional server components can be installed into WebSphere Application Server, where the Tivoli Dynamic Workload Broker server resides. They are:

- Tivoli Workload Scheduler Agent (Tivoli Dynamic Workload Broker component for integration with Tivoli Workload Scheduler — also known as TDWB/TWS Bridge agent)
- Enterprise Workload Manager (EWLM) enablement
- Tivoli Provisioning Manager (TPM) enablement
- IBM Change and Configuration Management Database (CCMDB) enablement

All the above listed components allow Tivoli Dynamic Workload Broker to integrate with other products. Installation of these components is optional.

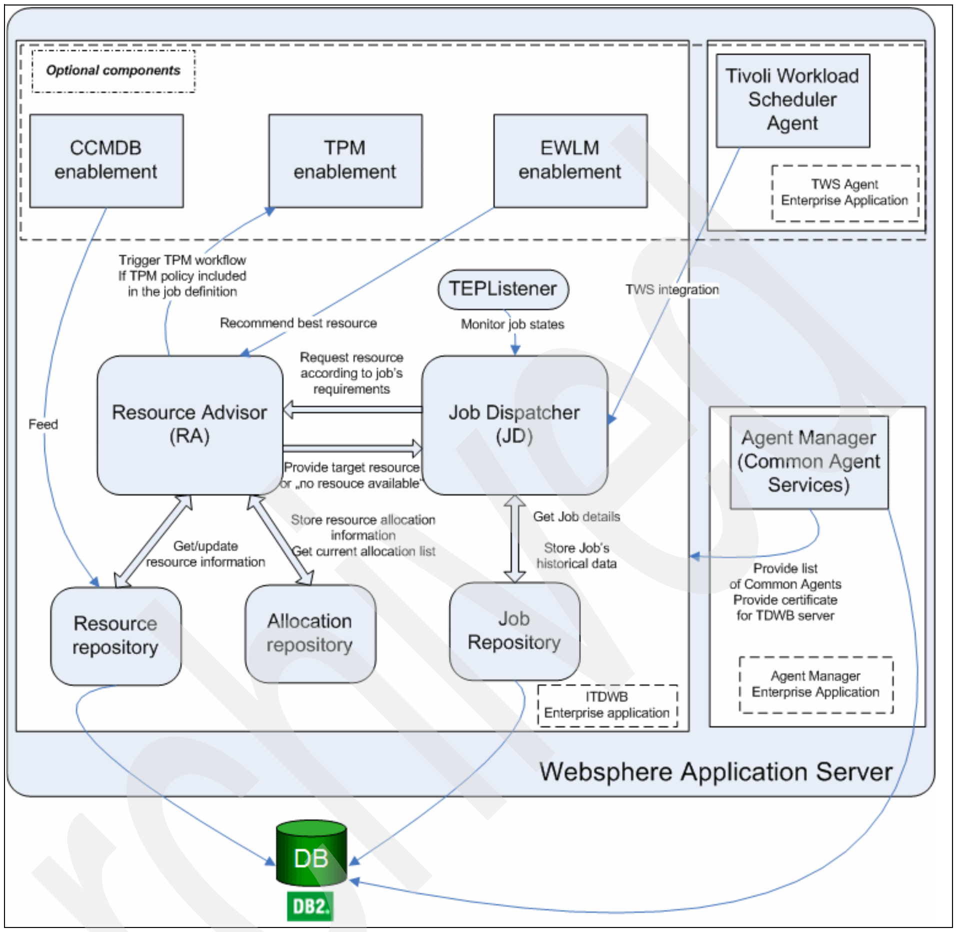
From the WebSphere Application Server perspective, only the Tivoli Workload Scheduler Agent (TDWB/TWS Bridge agent) is a separate enterprise application. The other integration components are installed directly into the ITDWB enterprise application. For more information about integrating Tivoli Dynamic Workload Broker with other products, refer to Chapter 8, "Integration with other IBM Tivoli products" on page 301.

By default the *TEPListener* component is also installed. TEPListener is responsible for the integration with IBM Tivoli Monitoring. This component is responsible for the logging of defined job states into a text file. This file can be observed by the IBM Tivoli Monitoring Universal Agent running on the same machine as the Tivoli Dynamic Workload Broker server. The job states can be displayed through the Tivoli Enteprise Portal. For more information about the integration of Tivoli Dynamic Workload Broker with IBM Tivoli Monitoring, see 8.4, “Integration with IBM Tivoli Monitoring” on page 325.

Tivoli Dynamic Workload Broker V 1.1 leverages IBM DB2® as its persistent data storage. DB2 can either reside on the same machine as Tivoli Dynamic Workload Broker server, or can be installed on a separate system. Then default name of the database created for Tivoli Dynamic Workload Broker server is *TDWB*. This name can be changed at installation time.

Note: Tivoli Dynamic Workload Broker V 1.1 can use only DB2, but in Tivoli Dynamic Workload Broker V 1.2 Oracle will also be supported. At the time of publishing this book the Tivoli Dynamic Workload Broker V 1.2. was not available. Every place that we mention DB2 as a relational database used by Tivoli Dynamic Workload Broker components, you can also add Oracle relational database, when related to Tivoli Dynamic Workload Broker V 1.2.

The schema of server components is shown in Figure 2-2.



Optional components
CCMDB enablement
TPM enablement
EWLM enablement
Tivoli Workload Scheduler Agent
TWS Agent Enterprise Application
Trigger TPM workflow If TPM policy included in the job definition
TEPListener
TWS integration
Recommend best resource
Monitor job states
Request resource according to job's requirements
Feed
Resource Advisor (RA)
Job Dispatcher (JD)
Agent Manager (Common Agent Services)
Provide target resource or „no resouce available"
Store resource allocation information
Get current allocation list
Get/update resource information
Get Job details
Store Job's historical data
Provide list of Common Agents Provide certificate for TDWB server
Resource repository
Allocation repository
Job Repository
ITDWB Enterprise application
Agent Manager Enterprise Application
Websphere Application Server
DB
DB2.

Figure 2-2 Tivoli Dynamic Workload Broker server architecture

The list of default physical locations is included in 2.9, “Physical location of Tivoli Dynamic Workload Broker’s components” on page 73.

Note: Tivoli Dynamic Workload Broker does not necessarily have to be installed onto a fresh WebSphere Application Server. It can be added to an existing WebSphere Application Server instance, but certain prerequisites must be met (for instance, WebSphere Application Server version and patch level). Before using an existing instance of WebSphere Application Server, consult the corresponding Tivoli Dynamic Workload Broker Installation Guide and Release Notes.

2.2.1 Resource Repository

The important information about the IT environment (from a job brokering perspective) is stored in the *Resource Repository*. It contains the list of available computers, defined resources, and a real-time performance of each managed resource. This data serves as input for the *Resource Advisor*.

Resource definitions and relations among various types of resources and resource groups are stored in tables within the DB2 database named TDWB.

It is also possible to import resources that are already maintained by the Change and Configuration Management Database (CCMDB). Tivoli Dynamic Workload Broker imports resources from the Change and Configuration Management Database and uses them as logical resources in its environment. See 8.2, "Integration with IBM Tivoli Change and Configuration Management Database (CCMDB)" on page 302 for more details about this topic.

2.2.2 Resource Advisor

Resource Advisor (RA) performs job resource matching. This means that it determines the best candidate for each job that is about to run. It uses complex evaluation methods to determine which currently available resource has the optimal capacity and meets all the necessary prerequisites for running a particular job. The main logic *what shall run where* is computed within this component.

In general, a heterogeneous set of resources is required to run a job. A job may need, for instance, a certain operating system, a file system, a network, and access to a database. A job may also request multiple instances of the same type of a resource. Only when all requirements are simultaneously satisfied can the job can be started.

The resource-matching process has the objective to select only the most capable resource where to send the job. The decision is made based on different factors:

- Availability of consumable resources - Resources are logically consumed by jobs while they are executing. The resource-matching process assigns the job execution to capable resources only when there are a sufficient amount of consumable resources available. During the resource-matching process the Resource Advisor considers that many different concurrent jobs may request a quantity of the same consumable attribute at the same time.
- Resource selection policies - The user can specify a resource selection policy that is applied to all capable targets. The user can specify additional criteria for the best fitting resources, such as CPU utilization, available physical

memory, and so on. By default, the resource selection policy of each job indicates the best resource that runs the minimum of jobs.

- Workload distribution process - The job with higher priority is privileged for the assignment of available resources. Prioritization is considered only when multiple jobs are competing for the same limited consumable resources.

2.2.3 Job Dispatcher

Job Dispatcher is responsible for managing requests for job submissions. It extracts resource requirements of the job, and passes them to the Resource Advisor. The resource requirements get converted into resource allocation requests. After identifying the best-fitting resource, Job Dispatcher submits the job to that resource. At this moment Job Dispatcher manages the job’s life cycle. Through its agent counterpart (called Job Execution Agent), the Job Dispatcher monitors the state of the job, gets the job’s status, and interacts with the job based on user inputs — either displays the job’s stdout or cancels the job on the user’s request.

2.2.4 Job Repository

The Job Repository keeps three types of data:

- Job definitions

  Job definitions that were created through:

  – Tivoli Dynamic Workload Broker Web Console

  – Job Brokering Definition Console

  – Imported through command-line interface from a JSDL file

  – Imported through the direct Web service invocation (custom application)

  Note: For more information about job definitions see 2.5.1, “Job definitions” on page 43.

- Job instances

  Job instances resulting from:

  – Jobs submitted from the Tivoli Dynamic Workload Broker Web Console

  – Jobs submitted to the Tivoli Dynamic Workload Broker from Tivoli Workload Scheduler

  – Jobs submitted to the Tivoli Dynamic Workload Broker through the command-line interface

- Historical data

  Historical data describing how the previous job instances ran.

The Job repository is physically represented as a couple of tables within the DB/2 database.

2.2.5 Allocation Repository

The Allocation Repository keeps data about the current resources' allocation. The data within Allocation Repository are maintained by the Resource Advisor.

The allocation repository is used to store the identifiers of the resources matched for a job. The quantities of consumed resources are kept in a cache. When the job terminates for any reason the allocation information is removed from the allocation repository.

2.3 Tivoli Dynamic Workload Broker agent

The Tivoli Dynamic Workload Broker agent is responsible for handling requests incoming from the server.

The Tivoli Dynamic Workload Broker agent runs as a subagent of the *Common Agent*. This term is explained in 2.4, "Common Agent Services" on page 38. The Tivoli Dynamic Workload Broker agent leverages the Common Agent Services architecture. For the understanding of the Tivoli Dynamic Workload Broker agent functionality it is not necessary to understand Common Agent Services at this point.

For an overview of Common Agent Services see 2.4, "Common Agent Services" on page 38. The information included in this chapter will help you to better understand how secure communication between the Tivoli Dynamic Workload Broker server and the Tivoli Dynamic Workload Broker Agent is performed.

The Tivoli Dynamic Workload Broker agent runs under the user account that was specified during installation. This account is also the default user ID, which is used for running the job on the target system when a job is submitted to the agent and no user ID and password were explicitly specified in the job definition. The only exception to this behavior is on the Windows platform, when the Tivoli Dynamic Workload Broker agent is installed under the *Local System* account. In this case, jobs without an explicit user ID and password definition run under the default administrator's account.

Note: This only applies to Windows since there is no concept of a local system account in UNIX. On UNIX platforms the agent runs under the nobody account, and jobs without explicit credential definitions are submitted under the root account.

The Tivoli Dynamic Workload Broker agent itself consists of two major components. Each of these major components splits into subcomponents, as described below.

2.3.1 Major agent components

Tivoli Dynamic Workload Broker agent is responsible for performing two major tasks: handling the jobs submitted to the agent and providing information about the hosting machine. There are two components that perform these tasks:

- Job Execution Agent - responsible for launching and cancelling jobs on the target system (system where agent resides) and tracking jobs' states. A detailed architecture is shown in Figure 2-3 on page 36.
- Resource Advisor Agent - responsible for gathering information about the machine where it runs. Maintains information about current machine's utilization (CPU, memory), available disk space, and so on. A detailed architecture is shown in Figure 2-4 on page 37.

2.3.2 Agent subcomponents

Both the Job Execution Agent and the Resource Advisor Agent are further split into second-level subcomponents:

- Job Execution Agent
  - Native Job Executor - launches jobs in the operating system's environments (batch files, scripts, executables)
  - J2EE Job Executor - launches jobs within J2EE
    - Enterprise Java Beans (EJB™) invocation
    - Java Message Services (JMS) posting

The schema of a Job Execution Agent and the corresponding server counterpart is shown in Figure 2-3.

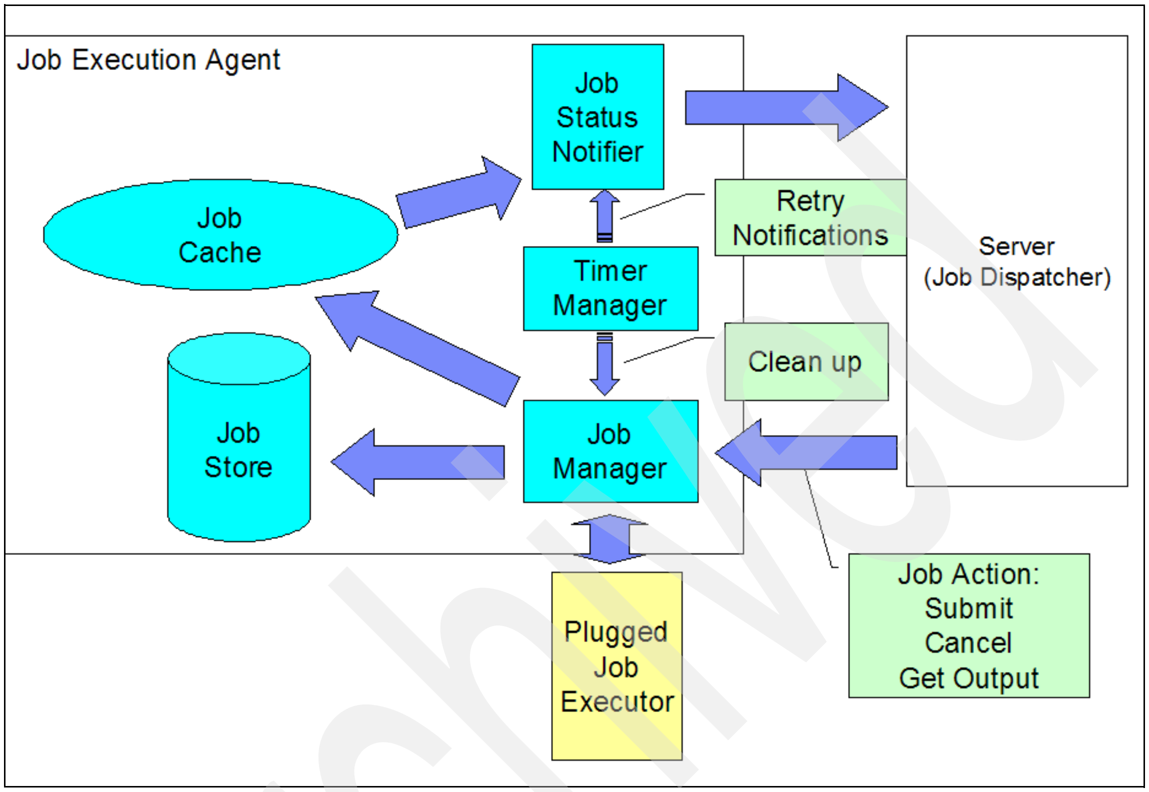


Figure 2-3 Job Execution Agent - architecture and interaction with server

- Resource Advisor Agent

  Base scanners - subcomponents responsible for scanning values from hosting operating system. Scanners collect data about:

  – Computer system
  – Operating system
  – Network addresses
  – File systems
  – Any defined logical resource

Scanners are based on *Common Inventory Technology* (CIT). The Common Inventory Technology originates from many sources (such as refactored code available of Inventory part of Tivoli Configuration Manager, Tivoli License Manager and others).

The features of the Common Inventory Technology:

- Is a common technology for hardware and software recognition and data collection
- Can be used by Tivoli products that need information about target system’s hardware and software configuration
- Permits sharing of recognition functionality and data among Tivoli products
- Provides a consistent view of hardware and software configuration information across Tivoli products
- Coordinates collection activities among Tivoli products
- Converges on a standard, common data model

The schema of the Resource Advisor Agent and the corresponding server counterpart is shown in Figure 2-4.

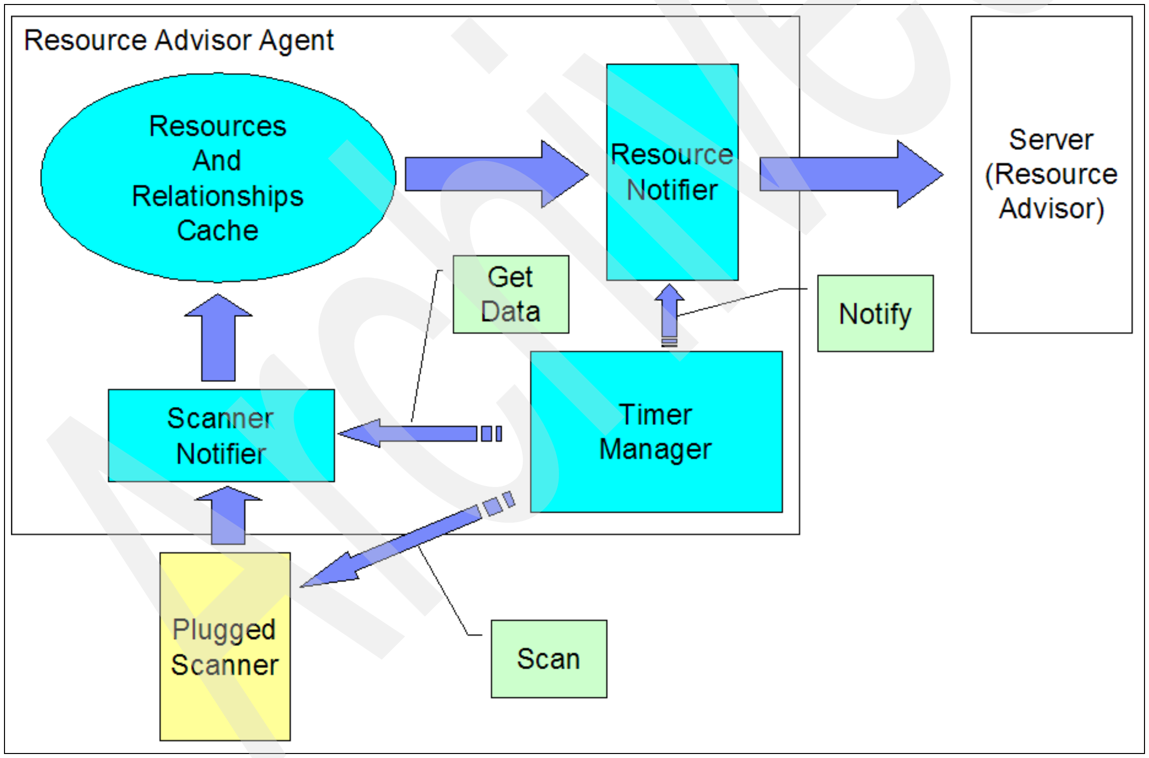


Figure 2-4 Resource Advisor Agent - architecture and interaction with server

2.4 Common Agent Services

This section describes the purpose and the architecture of the Common Agent Services.

Common Agent Services (CAS) is a shared infrastructure for managing the computer systems in your environment. Multiple products can share the Common Agent Services infrastructure if they require the same Common Agent Services version.

Common Agent Services consists of three main components:

- Common Agent - is the shared agent used by other application agents (such as Tivoli Dynamic Workload Broker agent). The common agent is in fact a *host* for the other application agents. Application agents are installed as a *subagents* of the Common Agent.
- Agent Manager - is a central point of Common Agent Services. This manages the Common Agents and integrates with other applications. This registers the other application servers as Resource Managers.
- Resource Manager - is a logical representation of the integrated application in the Agent Manager. From the Agent Manager's perspective, the Resource Manager represents the server of the integrated application. The Tivoli Dynamic Workload Broker server is one of possible Resource Managers that can be registered in the Agent Manager.

We explain these components more in detail in the following subsections.

Common Agent Services is just an infrastructure for other Tivoli products. By itself it does not perform any systems management tasks. It only offers a set of shared functions (such as shared libraries and secure communication) for the integrated product. Multiple Tivoli products leverage the Common Agent Services infrastructure. Be aware that different versions of each product may require the different version of Common Agent Services.

We provide a sample list of Tivoli products leveraging the Common Agent Services infrastructure:

- Tivoli Dynamic Workload Broker
- Tivoli Provisioning Manager for Software
- Total Storage Productivity Center for Fabric
- Total Storage Productivity Center for Data

Note: Do not confuse Common agent Services with Tivoli Management Environment® (TME®, also known as the Tivoli Framework). These two infrastructures are based on completely different code bases. Tivoli Common Agent is not the Tivoli Endpoint, and Agent Manager is not a Tivoli Management Region (TMR) server.

2.4.1 Agent Manager

Agent Manager is the central point of a Common Agent Services infrastructure. It acts as the integration point among the application servers (such as Tivoli Dynamic Workload Broker server) and *Common Agents.*

Agent Manager has following properties:

- Has its own Certification Authority for issuing certificates.
- Maintains the list of revoked certificates (CRL).
- Issues certificates for newly installed Common Agents. At registration time it creates the certificate with the private key and distributes it together with CRL to the new Common Agent.
- Issues certificates for new Resource Managers. At registration time it creates the certificate with the private key and distributes it to the new Resource Manager. The Tivoli Dynamic Workload Broker server must have a certificate signed by Agent Manager's certification authority to be able to communicate with Common Agents and thus with the Tivoli Dynamic Workload Broker subagents hosted by Common Agents.

Note: Agent Manager leverages DB2 as its persistent storage. Agent Manager can use the same DB2 as the Tivoli Dynamic Workload Broker server, or can use another DB2 instance. This depends on your deployment scenario. If you want the Agent Manager to use a different DB2 instance you need to install the Agent Manager separately from the Tivoli Dynamic Workload Broker server. If you install Agent Manager together with the Tivoli Dynamic Workload Broker, both Agent Manager and Tivoli Dynamic Workload Broker server will use the same DB2 instance.

In the Tivoli Dynamic Workload Broker V1.2 it will be possible to install an Agent Manager that uses an Oracle database.

2.4.2 Common Agent

Common Agent is a shared run time that runs on each managed system leveraging the *Common Agent Services* architecture. The Common Agent lets multiple management applications share resources when managing a system and provides them with remote deployment capability, management, and security. One of the possible managing applications is Tivoli Dynamic Workload Broker.

Common Agent is in fact a *host* for the other applications agents. An agent of a managing application is installed as a *subagent* of the Common Agent. Subagents use the Common Agent's shared libraries and certificate for establishing secure communication with the corresponding server counterpart.

Each Common Agent must first register to the Agent Manager to acquire the valid certificate. Only with the valid certificate is the Common Agent (and all of its subagents) able to communicate with other parties within the Common Agent Services infrastructure.

Be aware that different versions of each integrated application may require the different version of Common Agent.

2.4.3 Interaction between Tivoli Dynamic Workload Broker and Common Agent Services

The Tivoli Dynamic Workload Broker server is registered within the Agent Manager as a *Resource Manager*. Agent Manager issued a certificate for the Tivoli Dynamic Workload Broker server at the registration time. The Tivoli Dynamic Workload Broker server registers to Agent Manager when one of these two events occurs:

- User wants to see the list of agents known by the Agent Manager (through the Web Console interface)
- The Tivoli Dynamic Workload Broker server needs to connect to its agent

Tivoli Dynamic Workload Broker leverages the Common Agent Services infrastructure for two main purposes:

- The Common Agent Services infrastructure provides secure communication through SSL.

  The communication between the Tivoli Dynamic Workload Broker server and the Tivoli Dynamic Workload Broker Agent (which is in fact Common Agent's subagent) goes directly from the Tivoli Dynamic Workload Broker server to the Tivoli Dynamic Workload Broker agent. Agent Manager is not involved in the traffic between the Tivoli Dynamic Workload Broker server and Tivoli Dynamic Workload Broker agents.

Agent Manager was necessary at the registration time when it issued certificates either for the Tivoli Dynamic Workload Broker server (registered in Agent Manager as a *Resource Manager*) or for the Common Agent hosting the Tivoli Dynamic Workload Broker subagent. Both the Tivoli Dynamic Workload Broker server and the agent use certificates issued by the same certification authority and are able to perform mutual SSL handshakes and establish a secure communication channel.

After the SSL channel has been established the incoming traffic from the Tivoli Dynamic Workload Broker server goes directly to the listening port of Common Agent. Both Tivoli Dynamic Workload Broker subagents (Resource Advisor agent and Job Executor Agent) are exposing a *Web service* on that port.

- Based on Tivoli Dynamic Workload Broker server's request, Agent Manager provides a list of *all* registered Common Agents.

  The Tivoli Dynamic Workload Broker server can connect directly to the desired Common Agent and either install or uninstall a Tivoli Dynamic Workload Broker subagent.

Important note concerning agent lists: It is necessary to distinguish two different agent lists that are presented in the Tivoli Dynamic Workload Broker Web Console interface. Each list is achieved in different way, using different communication channels. Content of the list may also be different:

- Select **Tracking** → **Computers**. This view offers a list of Tivoli Dynamic Workload Broker agents.

  When a ITDWB Subagent is installed on the top of a Common Agent, it connects to the Tivoli Dynamic Workload Broker server. Its name is then stored in the list of known Tivoli Dynamic Workload Broker agents.

  In this case Agent Manager was not involved in the communication between Tivoli Dynamic Workload Broker server and agent.

- Select **Scheduling Environment** → **Agents**. This view offers a list of all Common Agents managed by the Agent Manager (including those ones that do not host a Tivoli Dynamic Workload Broker subagent).

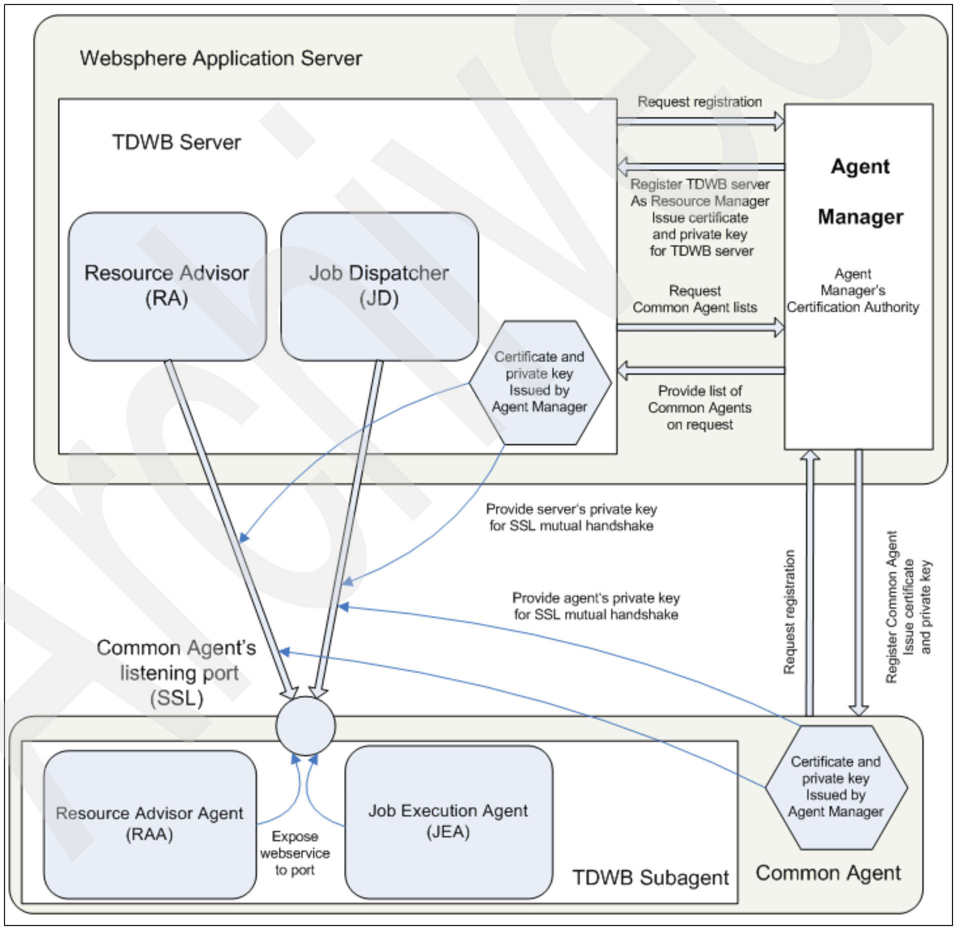
  In this case the Tivoli Dynamic Workload Broker contacted the Agent Manager and requested the list of installed Common Agents.

From the topological perspective the Agent Manager does not necessarily have to reside on the same machine as the main Tivoli Dynamic Workload Broker server. If Tivoli Dynamic Workload Broker is the first Tivoli product using the

Common Agent Services, the Agent Manager probably will be installed on the same machine as the other Tivoli Dynamic Workload Broker server components. However, in large environments, with multiple Tivoli products leveraging the Common Agent Services infrastructure, a separate server should be dedicated for the Agent Manager component.

For more information about the secure communication among the Tivoli Dynamic Workload Broker server and its agents refer to 2.7.1, “Encrypted communication” on page 52.

See Figure 2-5 for a clear view of the Tivoli Dynamic Workload Broker server integrated with Agent Manager, and thus leveraging Common Agent Services for communicating with its agents.



Websphere Application Server
TDWB Server
Resource Advisor (RA)
Job Dispatcher (JD)
Certificate and private key Issued by Agent Manager
Request registration
Register TDWB server As Resource Manager Issue certificate and private key for TDWB server
Request Common Agent lists
Provide list of Common Agents on request
Agent Manager
Agent Manager's Certification Authority
Provide server's private key for SSL mutual handshake
Provide agent's private key for SSL mutual handshake
Request registration
Register Common Agent Issue certificate and private key
Common Agent's listening port (SSL)
Resource Advisor Agent (RAA)
Expose webservice to port
Job Execution Agent (JEA)
TDWB Subagent
Certificate and private key Issued by Agent Manager
Common Agent

Figure 2-5 Communication between Tivoli Dynamic Workload Broker server and agent

2.5 Job and resource definitions

In this section we describe the format that is used for storing the job and resource definitions. We also list the possibilities of how the definitions can be created.

Tivoli Dynamic Workload Broker job definitions and resource definitions are both stored in the Tivoli Dynamic Workload Broker repositories. However, the way in which they can be created and modified is different.

The persistent storage for both repositories is IBM database DB/2. All the data are stored in DB/2 tables within the database dedicated for Tivoli Dynamic Workload Broker usage. The default name of this database is TDWB.

2.5.1 Job definitions

Tivoli Dynamic Workload Broker uses a special language for defining jobs. This language is based on XML and its name is *Job Submission Description Language (JSDL)*.

Each job definition is stored in the *Job Repository*. When saving the job definition to the Job Repository, a validation is run against the current JSDL definition. If a definition is not valid (it does not meet the JSDL schema), it is not saved until the errors are corrected.

Job definitions can be created using following tools/interfaces:

- Tivoli Dynamic Workload Broker Web Console
- Job Brokering Definition Console
- Imported through command-line interface from a JSDL file
- Imported through direct Web service invocation (custom application)

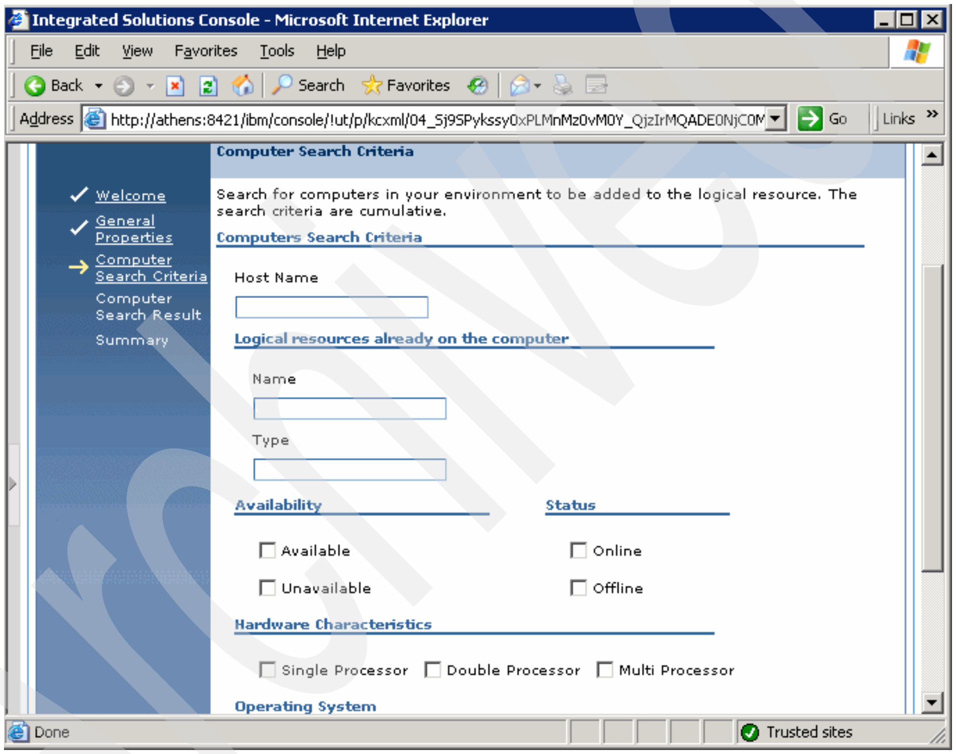
The Tivoli Dynamic Workload Broker Web Console offers you the possibility to create the job definition directly on the server, but does not provide any sophisticated tool for this task. It requires comprehensive knowledge of JSDL schema to be able to create correct job definitions from scratch.

Instead of writing the JSDL definitions manually, you can use a graphical tool called Job Brokering Definition Console (JBDC). JBDC offers an intuitive user interface for creating the job definitions. It also performs validation of the job definition against the Job Submission Description Language (JSDL) schema, so you will be informed of any possible error that you made during the job definition. For more information about Job Brokering Definition Console, refer to 2.6.3, “Job Brokering Definition Console” on page 49.

2.5.2 Resource definitions

Resource definitions can be created and modified through the Tivoli Dynamic Workload Broker Web Console in a much more fashionable way than job definitions. Web Console offers the *Resource wizards*, which perform step-by-step resource definition.

All the work necessary for manipulating with resource definition can be done through the native Tivoli Dynamic Workload Broker Web Console. An example is shown in Figure 2-6.


Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://athens:8421/ibm/console/!ut/p/kcxml/04_Sj9SPykssy0xPLMnMz0vM0Y_QjzIrMQADE0NjC0M
Go Links
Welcome
General Properties
Computer Search Criteria
Computer Search Result
Summary
Done
Trusted sites

Computer Search Criteria

Search for computers in your environment to be added to the logical resource. The search criteria are cumulative.

Computers Search Criteria

Host Name

Logical resources already on the computer

Name

Type

Availability

- Available
- Unavailable

Status

- Online
- Offline

Hardware Characteristics

- Single Processor
- Double Processor
- Multi Processor

Operating System

Figure 2-6 Resource wizard in Tivoli Dynamic Workload Broker Web Console

2.6 Tivoli Dynamic Workload Broker User interfaces

In this section we list the user interfaces provided with Tivoli Dynamic Workload Broker and describe their capabilities.

Tivoli Dynamic Workload Broker offers a Web interface, command-line interface, and graphical tool for defining the Tivoli Dynamic Workload Broker jobs. We describe all of them in following sections.

All the user interfaces use the Web services interface provided by the Tivoli Dynamic Workload Broker server (the ITDWB enterprise application running in the WebSphere Application Server).

2.6.1 Tivoli Dynamic Workload Broker Web Console

The Web interface of Tivoli Dynamic Workload Broker is called Tivoli Dynamic Workload Broker *Web Console.*

Tivoli Dynamic Workload Broker leverages *Integrated Solutions Console (ISC)* and puts its Web Console into the ISC portal solution.

In fact, the Integrated Solutions Console is a common interface for multiple products, such as Tivoli Storage Manager (TSM), Tivoli Workload Scheduler (TWS) and so on.

The Integrated Solutions Console should be installed on a different machine from the Tivoli Dynamic Workload Broker server. It is also runs within separated WebSphere Application Server. For more details about default installation paths see 2.9, "Physical location of Tivoli Dynamic Workload Broker's components" on page 73.

Note: The Integrated Solutions Console cannot be installed into the same WebSphere Application Server because the Integrated Solutions Console shipped with Tivoli Dynamic Workload Broker server V 1.1 uses an embedded WebSphere Application Server instance.

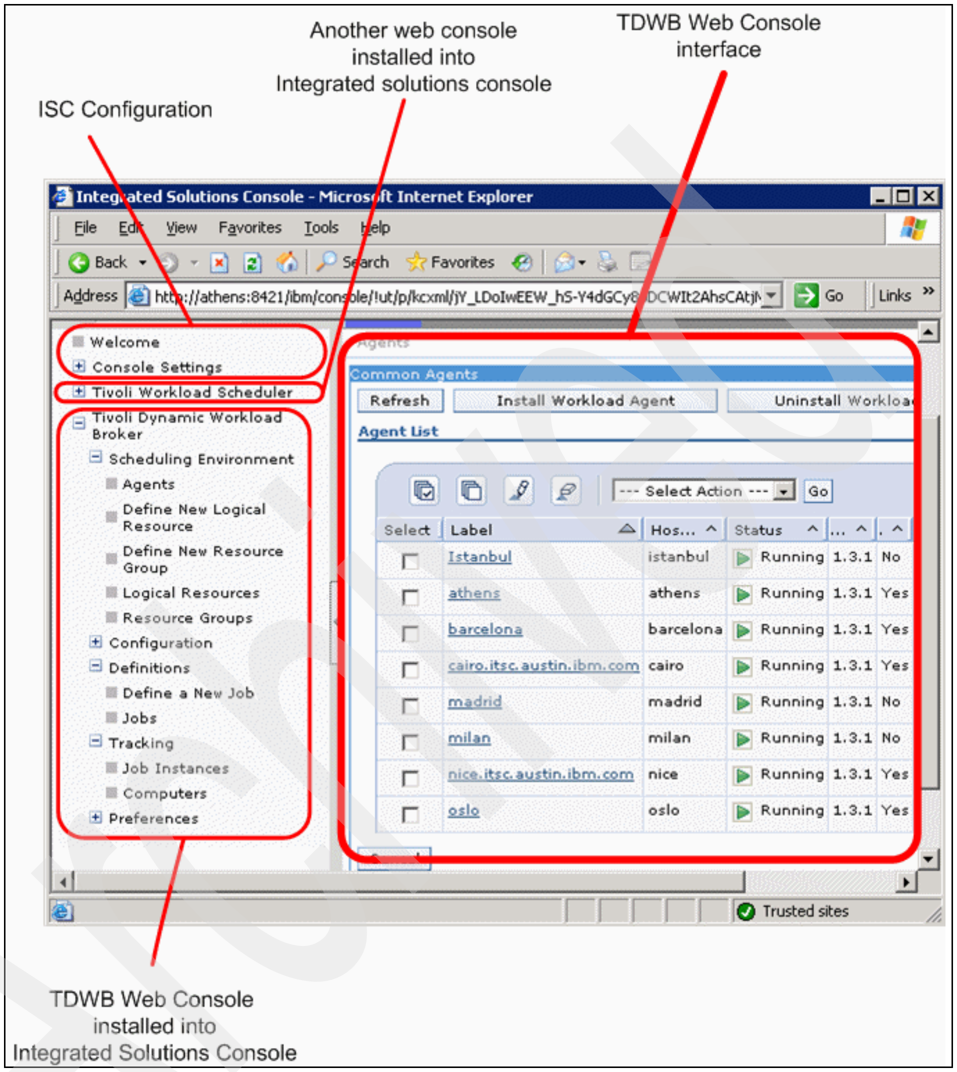
This statement does not mean that you cannot install the Integrated Solutions Console on the same machine as the Tivoli Dynamic Workload Broker server. We just emphasize the fact that the Integrated Solutions Console (ISC) J2EE application cannot be installed onto the same WebSphere Application Server instance that hosts the Tivoli Dynamic Workload Broker server (the ITDWB J2EE application).

Tivoli Dynamic Workload Broker Web Console provides a Web interface for managing the Tivoli Dynamic Workload Broker environment. It offers, for instance:

- Defining computers and logical resources
- Editing job definitions
- Submitting and monitoring jobs
- Recovering failing jobs or resources

It is possible to connect to the Tivoli Dynamic Workload Broker Web Console with any supported browser.

You can see an example of the Web Console in Figure 2-7.


Another web console installed into Integrated solutions console
TDWB Web Console interface
ISC Configuration
Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://athens:8421/ibm/console/!ut/p/kcxml/jY_LDoIwEEW_hS-Y4dGCy8DCWIt2AhsCAtjN Go Links
Welcome
Console Settings
Tivoli Workload Scheduler
Tivoli Dynamic Workload Broker
Scheduling Environment
Agents
Define New Logical Resource
Define New Resource Group
Logical Resources
Resource Groups
Configuration
Definitions
Define a New Job
Jobs
Tracking
Job Instances
Computers
Preferences
Agents
Common Agents
Refresh
Install Workload Agent
Uninstall Workloa
Agent List
--- Select Action --- Go
Trusted sites
TDWB Web Console installed into Integrated Solutions Console

Select	Label	Hos...	Status	...	.
☐	Istanbul	istanbul	Running	1.3.1	No
☐	athens	athens	Running	1.3.1	Yes
☐	barcelona	barcelona	Running	1.3.1	Yes
☐	cairo.itsc.austin.ibm.com	cairo	Running	1.3.1	Yes
☐	madrid	madrid	Running	1.3.1	No
☐	milan	milan	Running	1.3.1	No
☐	nice.itsc.austin.ibm.com	nice	Running	1.3.1	Yes
☐	oslo	oslo	Running	1.3.1	Yes

Figure 2-7 Tivoli Dynamic Workload Broker Web console leveraging ISC

2.6.2 Command-line interface

Tivoli Dynamic Workload Broker also provides a *command-line interface* (CLI) that offers similar functionality as a graphical Web Console. CLI is useful for those users who prefer to administer the environment from the command line. CLI also offers wider possibilities for scripting of more complicated tasks.

The command-line interface (CLI) allows the Tivoli Dynamic Workload Broker user to perform all the essential operations necessary for managing the jobs. By using a CLI, a Tivoli Dynamic Workload Broker user can also put jobs into the archive database tables.

A command-line interface offers following functionality:

- Managing job definitions
- Submitting jobs
- Getting job details
- Cancelling jobs
- Querying jobs
- Archiving database tables

We provide simple mapping among these operations and their corresponding commands below.

Example 2-1 shows the job submission from the command line.

Example 2-1 Submitting a job from command-line interface

```
C:\Documents and Settings\Administrator>jobsubmit -jdname first
Call Job Dispatcher to submit the job.
Success returned from Job Dispatcher
The job d5ac970a-588a-3eaf-bfcf-b13d8d62b0b3 submitted successfully
```

The default path of command line binaries is the /bin subdirectory of the Tivoli Dynamic Workload Broker installation directory:

- On Windows - C:\Program Files\ITDWB\Server\bin
- On UNIX platforms - /opt/IBM/ITDWB/Server/bin

Before you can launch any of the CLI executables, you must source the Tivoli Dynamic Workload Broker environment:

- On Windows C:\Program Files\ITDWB\Server\bin\tdwb_env.bat
- On UNIX . /opt/IBM/ITDWB/Server/bin/tdwb_env.sh

Table 2-1 describes mapping among Tivoli Dynamic Workload Broker operations and their corresponding commands.

Table 2-1 Command Line Interface - command list

Task	Related command
Manage job definitions.	**`jobstore`**
Job submission.	**`jobsubmit`**
Getting single job details.	**`jobdetails`**
Cancel job.	**`jobcancel`**
Query jobs.	**`jobquery`**
Archive database tables.	**`movehistorydata`**

All of the CLI executables can be run only on the Tivoli Dynamic Workload Broker server. There is currently no deployment scenario available that allows you to install Tivoli Dynamic Workload Broker CLI on a separate machine and issue commands remotely.

2.6.3 Job Brokering Definition Console

The Job Brokering Definition Console is a graphical tool that serves as a user-friendly interface for creating Tivoli Dynamic Workload Broker job brokering definitions.

The Job Brokering Definition Console also servers as a conversion tool for transforming the Tivoli Workload Scheduler (TWS) jobs into Tivoli Dynamic Workload Broker job brokering definitions.

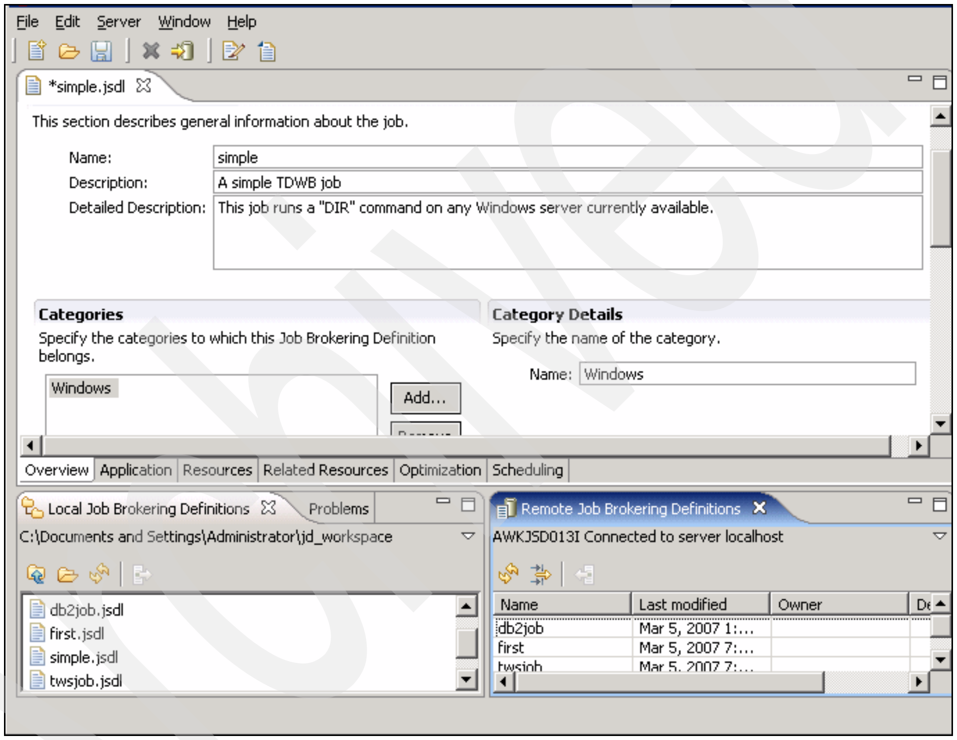
The Job Brokering Definition Console is typically installed on the workstations of Tivoli Dynamic Workload Broker administrators developing the job definitions. The Job Brokering Definition Console requires a TCP/IP connection to the Tivoli Dynamic Workload Broker server.

Creating and modifying Tivoli Dynamic Workload Broker job definitions

Tivoli Dynamic Workload Broker job definitions are written in an XML-based language called *Job Submission Description Language (JSDL)*. Unlike the resources, there is no wizard for job definitions in the Tivoli Dynamic Workload Broker Web Console. The Web Console offers only a plain text editor, where you have to write the whole JSDL code by yourself.

Creating a JSDL definition in a plain text editor could be a hard task. First, you must know the syntax Job Submission Description Language, and second, you must avoid any error and strictly follow the JSDL schema. Both of these tasks can be very time consuming. In most cases the direct access to JSDL definitions is suitable only for minor definition changes, but not for creating the definition from scratch.

Job Brokering Definition Console (JBDC) is a tool that does this all for you. It provides a user interface for an easy and intuitive job definitions creation. You can see a snapshot of the Job Brokering Definition Console in Figure 2-8.



This section describes general information about the job.

Name: simple

Description: A simple TDWB job

Detailed Description: This job runs a "DIR" command on any Windows server currently available.

Categories

Specify the categories to which this Job Brokering Definition belongs.

Windows

Category Details

Specify the name of the category.

Name: Windows

C:\Documents and Settings\Administrator\jd_workspace

db2job.jsdl
first.jsdl
simple.jsdl
twsjob.jsdl

AWKJSD013I Connected to server localhost

Name	Last modified	Owner	De
db2job	Mar 5, 2007 1:...		
first	Mar 5, 2007 7:...		
twsjob	Mar 5, 2007 7:...		

Figure 2-8 Job Brokering Definition Console interface

Note: Job Brokering Definition Console can only create *job definitions*. If you want to create resource definitions, you must use Tivoli Dynamic Workload Broker Web Console wizards.

Job Brokering Definition Console can be installed on a remote workstation. It can work with job definitions stored in two repositories:

- Local file system
- Server job repository

You can download or upload the job definitions between your local and server repository. Any new job definition made by JBDC is stored locally first. It is not visible on the server until you upload this definition.

A different approach should be used when *modifying* the job definitions. Even if you can modify your local job definition, and after that upload it to the server, that is not the best way. The best practice is to use the server copy. The reason for that is simple: job definitions can be modified also through the Tivoli Dynamic Workload Broker Web Console. If somebody did that, and you overwrite the job definition with your local copy, changes made previously to the server copy will be lost.

Another thing that you have to keep in mind is the fact that JBDC does not have a direct link to Tivoli Dynamic Workload Broker resources. This means that you must know the exact resource name when defining a resource dependency for a job. JBDC offers you several boxes for filling in resources that are necessary for running a job, but it does not provide any list of already defined resources. If you want to fill in any resource, you must go to the Tivoli Dynamic Workload Broker Web Console first, read the resource name there, go back to the Job Brokering Definition Console, and write resource name there.

Converting Tivoli Workload Scheduler job definitions into Tivoli Dynamic Workload Broker job brokering definitions

Job Brokering Definition Console is also a transformation tool used for converting Tivoli Workload Scheduler job definitions into Tivoli Dynamic Workload Broker job brokering definitions. The detailed step-by-step mechanism is described in the 5.1, “Tivoli Workload Scheduler migration to Tivoli Dynamic Workload Broker” on page 176.

2.7 Security features

In this section we describe the security features of Tivoli Dynamic Workload Broker broker.

Software solutions used for job workload management have the authority to launch jobs across many machines and platforms across the IT environment. A misuse of a such software tools could lead to severe security threats, such as:

- Unauthorized job management
  - Submitting of jobs
  - Cancelling of jobs
  - Providing the jobs with inappropriate parameters
- Sniffing of job outputs
- Sniffing logon credentials sent with the jobs

To reduce the security risks, the following features are included in Tivoli Dynamic Workload Broker:

- Encrypted communication
- Firewall support
- Authentication mechanism
- Authorization roles

All the above-mentioned features are described in following chapters.

2.7.1 Encrypted communication

In this section we describe how the Tivoli Dynamic Workload Broker server communicates with external components. We describe in detail two different communication *networks* that Tivoli Dynamic Workload Broker uses.

Tivoli Dynamic Workload Broker uses *Web services* as a communication mechanism among its components. The Tivoli Dynamic Workload Broker server accepts incoming Web services calls either from clients or from agents. Also, the agents accept Web services calls incoming from the server.

Tivoli Dynamic Workload Broker uses either HTTP or HTTPS (HTTP over SSL) protocols as the transport layer. This chapter describes how the secure channels can be established among the Tivoli Dynamic Workload Broker components.

Two communication networks

From the topological point of view we can determine two different communication networks among the Tivoli Dynamic Workload Broker server and external components:

- Communication with clients
  - Web Console running in the Integrated Solutions Console
  - Job Brokering Definition Console
  - Command-line interface
  - Custom external applications that are using Web service calls
- Communication with managed agents
  - Tivoli Dynamic Workload Broker subagents hosted by Common Agents

You can see both communication networks in Figure 2-9.

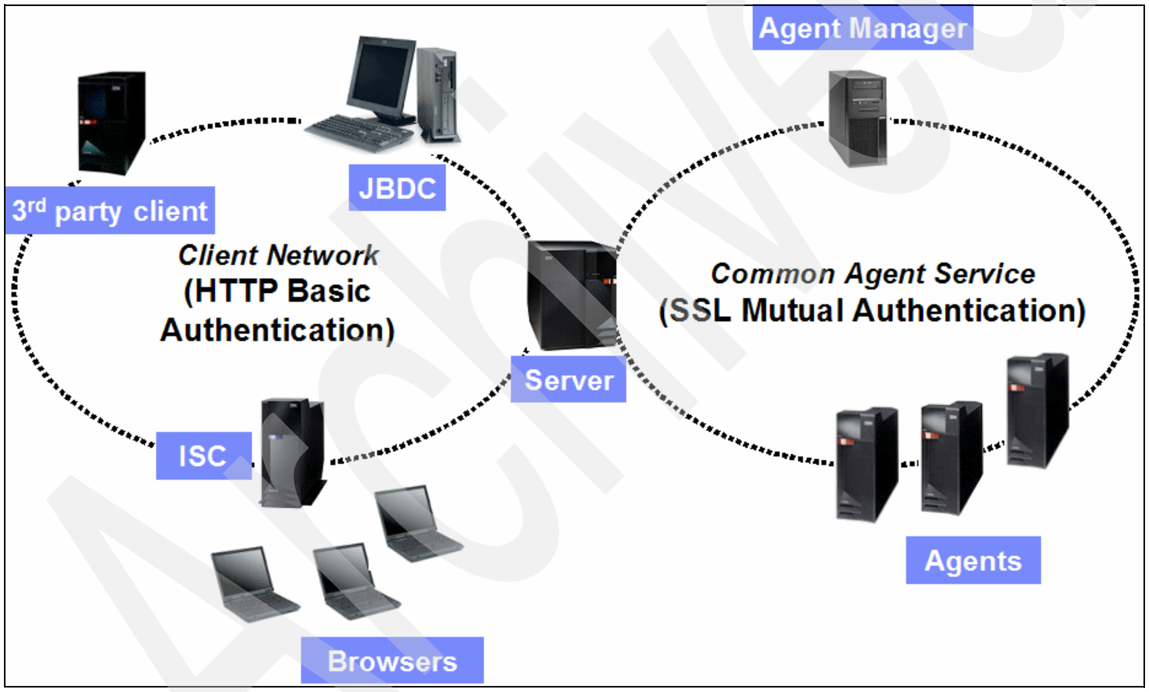


Figure 2-9 Communication networks

We describe both of the communication networks in the following sections.

Server → Agent communication

In this section we describe what mechanism is used for establishing trusted communication among Tivoli Dynamic Workload Broker server and Tivoli Dynamic Workload Broker agents.

The Tivoli Dynamic Workload Broker server leverages a Common Agent infrastructure for communication with its agents.

Common Agent Services consists of the following parties:

- Agent Manager - central point of Common Agent Services. This has its own Certification Authority, which is used for issuing certificates to Common Agents and Resource Managers.
- Common Agents - agents installed on each managed system. Tivoli Dynamic Workload Broker agents are installed as *subagents* of Common Agents. Each Common Agent has a certificate issued by Agent Manager.
- Resource Manager - the server part of the application that is using the Common Agent Services infrastructure. From Agent Manager's perspective the Tivoli Dynamic Workload Broker server is a *Resource Manager*.

There are different events that invoke the registration either of a Tivoli Dynamic Workload Broker server or a Tivoli Dynamic Workload Broker agent:

- Tivoli Dynamic Workload Broker servers register to Agent Manager when one of these two events occurs:
  - User wants to see the list of agents known by the Agent Manager (through the Web Console interface)
  - The Tivoli Dynamic Workload Broker server needs to connect to an agent
- Tivoli Dynamic Workload Broker agent registers in the installation time (if not installed in disconnected mode)

Each party registered in the Common Agent Services infrastructure has a pair made up of a unique *certificate* and a *private key*. Both certificate and private key were generated and signed by Agent Manager's Certification Authority during registration. Together with the key-pair, the Certification Authority's certificate was provided to each Agent and Resource Manager.

These certificates are used for a mutual SSL handshake when establishing the secure channel between Tivoli Dynamic Workload Broker server and Tivoli Dynamic Workload Broker Agent. The term *mutual* means that *both* parties *must* use their certificates and private keys during SSL handshake and thus prove their identity. Each party trusts the other because their certificates have been issued by the same certification authority.

Note: Each certificate issued by the Agent Manager is unique, but all the certificates are of the same type. The certificates issued for the Resource Manager (Tivoli Dynamic Workload Broker server) and for a Common Agent (hosting Tivoli Dynamic Workload Broker subagent) are each unique, but it is not possible to distinguish whether the certificate was issued for the a *managing server* or a *managed agent*. Due to this, it is possible to establish a trusted secure communication not only between server and agent, but also between two agents. Theoretically, a Tivoli Dynamic Workload Broker Agent could be able to accept instructions from any other Common Agent that would use correct Web services calls.

The communication between the Tivoli Dynamic Workload Broker server and Tivoli Dynamic Workload Broker Agent (the Common Agent's subagent) goes directly from the Tivoli Dynamic Workload Broker server to the Tivoli Dynamic Workload Broker agent and is not routed through the Agent Manager. The Agent Manager played its most important part during the registration time when it issued certificates.

Important: Because of the concept of Common Agent Services, the communication between the Tivoli Dynamic Workload Broker server and the Tivoli Dynamic Workload Broker agent is always *encrypted*. This is a must and encrypted communication cannot be switched off.

For additional information about Common Agent Services infrastructure, see 2.4, "Common Agent Services" on page 38.

Client → Server communication

In this section we describe what mechanism is used for establishing trusted communication among the Tivoli Dynamic Workload Broker server and its clients.

First we introduce the out-of-box provided user interfaces:

- Web Console running in Integrated Solutions Console
- Command-line interface
- Job Brokering Definition Console

The same mechanism as the user interfaces leverages two additional clients:

- Tivoli Workload Scheduler Agent
- External applications using API interface based on Web services technology

All Tivoli Dynamic Workload Broker clients use a Web services interface when communicating with the Tivoli Dynamic Workload Broker server. The transport protocol used for the communication can be either HTTP or HTTPS.

Both HTTP and HTTPS protocols are enabled by default on the server side. This means that the Tivoli Dynamic Workload Broker server accepts Web services calls that were delivered using either secure or non-secure protocols. Depending on the client's configuration, you may use either HTTP or HTTPS. All the clients are configured to use the HTTP (unecrypted) protocol by default. The following configuration steps must be done to change the configuration of clients for enforcing HTTPS:

- Job Brokering Definition Console - When defining the connection to the Tivoli Dynamic Workload Broker server you can choose whether you want to use a secure connection.
- Integrated Solutions Console - In the view **Tivoli Dynamic Workload Broker** → **Configuration** → **Server Connections** you can select whether you want to use the secure connection.
- Command-line interface - uses by HTTP default. By editing the CLIConfig.properties file you can select whether you want to use the secure connection.

Note: In a production environment we strongly recommend using HTTPS on all clients, because without encryption goes all the communication in clear text.

A secure channel between a server and its client is established using the out-of-box populated truststores.

Each party (server and clients) already has out-of-box keystores populated as follows:

- Client side:
  - TDWBClientKeyFile.jks - contains the client's private key
  - TDWBClientTrustFile.jks - contains the server's certificate
- Server side:
  - TDWBServerKeyFile.jks - contains the server's private key
  - TDWBServerTrustFile.jks - contains the client's certificate

  For physical locations of those files either on the client or server side, see 2.9.3, "Location of certificates and private keys" on page 77.

As shown in the list above, each party has locally stored its own private key and the certificate of the counterpart. Thus, everything necessary for the SSL handshake is prepared on both sides. By default only a server-side SSL handshake is implemented.

Summary of default certificates used in secure communication

In this section we provide the list of all possible certificates and private keys that are by default used by the Tivoli Dynamic Workload Broker.

As we have stated before, Tivoli Dynamic Workload Broker uses two types of communication networks. We list the certificates and private keys for both of them:

- Certificates and private keys used in client network:
  - Client's private key and certificate: For each client there is the same key pair (private/public). A client's private key is stored on the client, and client's certificate (which includes client's public key) is stored in the server's truststore on the server.

    By default the same client's private key is stored on all clients and the client's certificate stored on the server is common for all clients. The private and public keys are already included in the installation bundles. They are not generated at installation time.
  - Server's private key and certificate - A server's private key is stored on the server. A server's certificate (which includes a server's public key) is stored on each client.

    By default the same private key is stored on any installed Tivoli Dynamic Workload Broker server and the same server's certificate is stored on each installed client. The private and public keys are already included in the installation bundles. They are not generated at installation.

    The default certificates and private keys are used only for a SSL handshake. They provide the confidentiality and integrity features (nobody can read or change transferred data), but do not provide authentication (they do not ensure the identity of any client). However, you may use your own certificates that will correspond to each particular system.
- Certificates and private keys used in agent network
  - Agent's private key and certificate - They are both generated by Agent Manager during the registration time. Each Common Agent registers to Agent Manager and provides the common registration password. Agent Manager then generates the certificate and private key for that agent. They are always unique. The agent's private key is stored on the agent, and the agent's certificate (which includes the agent's public key) is stored in the Agent Manager's truststore. Also, the certificate of Agent Manager's Certification Authority is transferred to the agent during registration.
  - Server's private key and certificate - generated by Agent Manager at registration of the Tivoli Dynamic Workload Broker server to the Agent

Manager. During registration the Tivoli Dynamic Workload Broker server must provide the correct registration password. Agent Manager then generates a certificate and private key for the Tivoli Dynamic Workload Broker server and registers it as a Resource Manager. This certificate is always unique. Also, the certificate of Agent Manager's Certification Authority is transferred to the server (Resource Manager) during registration.

Figure 2-10 shows the communication of Tivoli Dynamic Workload Broker server with its agents and clients. It also shows the certificates involved in establishing secure communication.

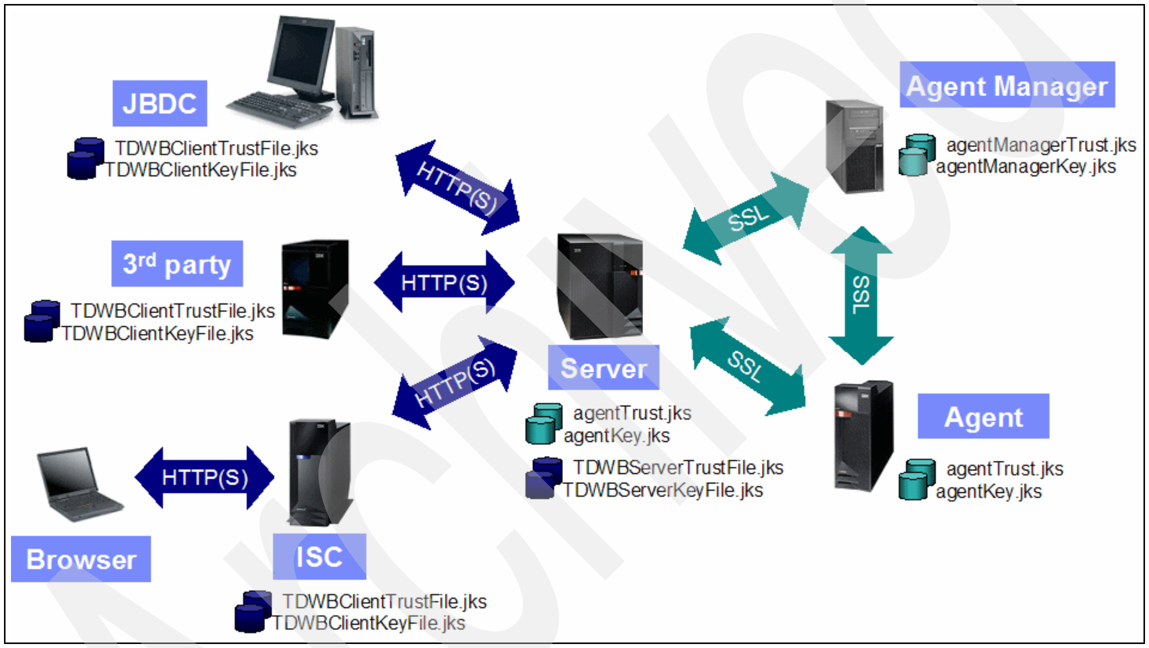


Figure 2-10 Communication networks and used certificates

2.7.2 Firewall support

Tivoli Dynamic Workload Broker uses strictly defined ports for inbound communication. When placed in firewall environment, it is sufficient to specify corresponding *allow* attributes for the listening ports used by Tivoli Dynamic Workload Broker components. See Figure 2-11 to get an idea of how the Tivoli Dynamic Workload Broker traffic passes through the zone boundaries.

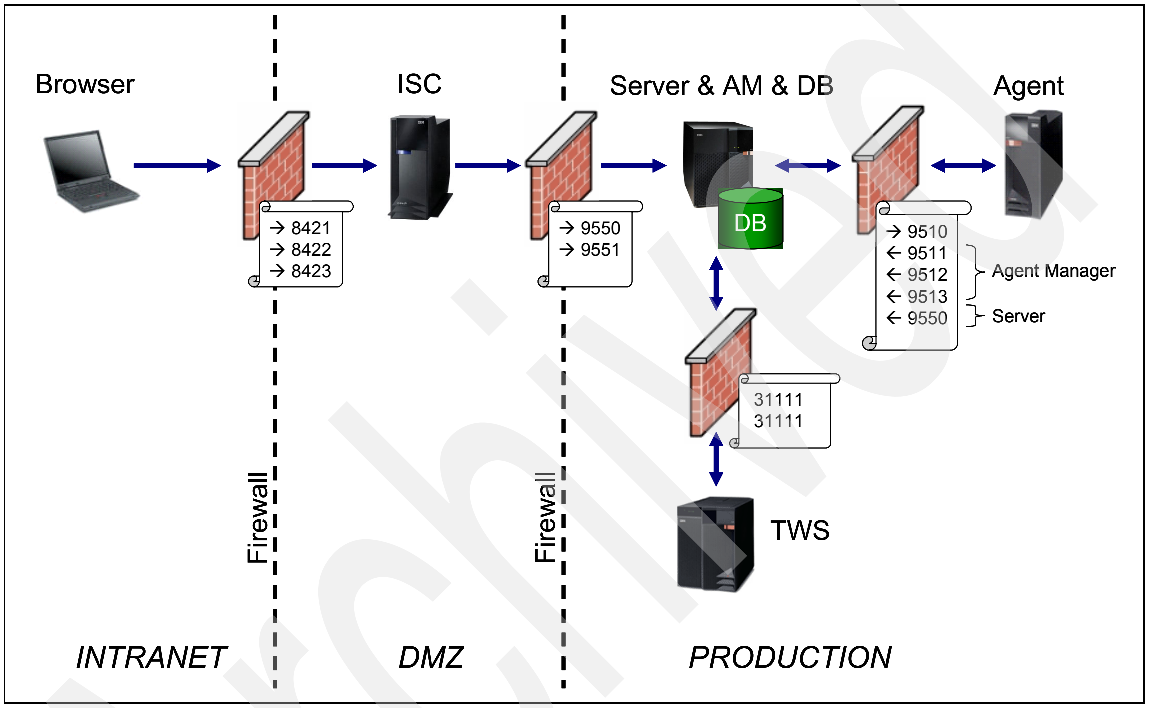


Figure 2-11 Sample - interaction with firewalls

For a detailed list of ports used by Tivoli Dynamic Workload Broker components refer to Appendix B, “Default ports used by Tivoli Dynamic Workload Broker” on page 657.

2.7.3 Authentication mechanism

In previous sections we explained the two different communication networks:

- Client network
- Agent network

In each of these networks a different authentication mechanism is used. We explain them more in detail in the following sections.

Authentication in agent network

Authentication in the agent's network is assured by the mechanism of Common Agent Services. Each party registered in the Common Agent Services has its unique certificate, which proves the identity of each party performing the mutual SSL handshake (establishing the secure connection).

No additional authentication is implemented. Proving the identity using the certificate is sufficient.

Authentication in client network

When speaking about the authentication mechanism, we must distinguish the data flows among various clients and Tivoli Dynamic Workload Broker server. In fact, we can separate the clients into two categories:

- Clients directly calling the listening server port on which Tivoli Dynamic Workload Broker server exposes Web services. Clients that communicate with the server directly are:
  - Job Brokering Definition Console
  - Command Line Interface
  - Any application using Web services interface (such as Tivoli Workload Scheduler Agent or any custom application)
- Web browser connecting to Tivoli Dynamic Workload Broker Web Console. The browser points to the Integrated Solutions Console portal first. Tivoli Dynamic Workload Broker Web Console running within the portal then handles the request and calls the listening server port, on which Tivoli Dynamic Workload Broker server exposes Web services.

Indenpendently on the client's category, the authentication on Tivoli Dynamic Workload Broker server is dependent on WebSphere Application Server's settings. Authentication is enforced when WebSphere Application Server has Global Security enabled. By default WebSphere Application Server does not have Global Security switched on.

A detailed description of how to switch on the authentication for Tivoli Dynamic Workload Broker server is included in 4.2.4, "Credentials for job definitions" on page 152.

Direct client → Server authentication

In this section we describe the mechanism of authentication in direct client - server communication and how this authentication can be turned on.

The Tivoli Dynamic Workload Broker in installed into WebSphere Application Server and leverages the WebSphere Application Server's authentication mechanism. Tivoli Dynamic Workload Broker does not implement any internal authentication infrastructure for communication with clients.

The authentication mechanism is enabled on the Tivoli Dynamic Workload Broker server when the *Global Security* of the WebSphere Application Server is enabled.

Important: Global Security for the WebSphere Application Server V 6.0 is disabled by default. It can be enabled by the WebSphere Administrative Console. Be aware that an Enable J2EE security check box will be preselected if you enable the Global Security. If you enable the Global Security, make sure that you have deselected the Enable J2EE security check box prior to submitting the request. The J2EE security *must not* be enabled on the WebSphere Application Server V 6.0 hosting Tivoli Dynamic Workload Broker server.

Depending on the WebSphere Application Server configuration, the user authentication is performed either against local operating systems user authority, or against a defined LDAP. WebSphere Application Server by itself does not contain any user definitions and credential vaults.

When the Global Security is turned on, each client must provide a user ID and password. These credentials are transported within the HTTP header because a HTTP Basic Authentication is used. Every time a request is transferred from client to server, it carries in the HTTP header the credentials. WebSphere Application Server takes care of the authentication then (either using operating systems authority or by checking the credentials with the LDAP).

Note: In this section we do not discuss *encrypted communication*, but about *authentication*. We stress this fact because we want to avoid a confusion. There is a difference between these terms. For instance, you may use a secure channel that was established between client and server using the server certificate for the SSL handshake. But if the WebSphere Application Server does not have Global Security turned on, you still do not have an authentication mechanism in place, even if you use certificates and SSL.

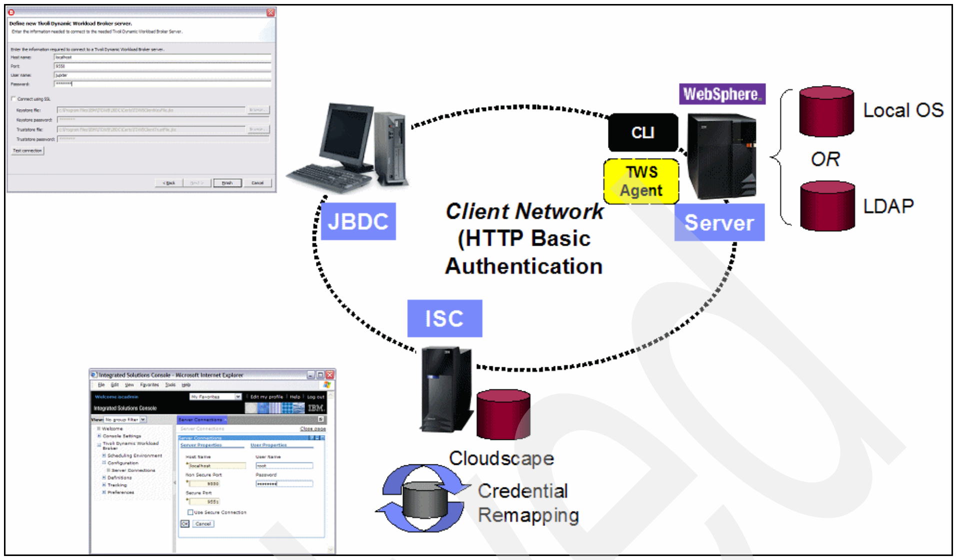


Figure 2-12 Client-Server authentication

Browser → ISC → Web Console → server authentication

In this section we describe the mechanism of authentication when a Tivoli Dynamic Workload Broker Web Console is used for communication with a server.

In this case we must split the authentication into two steps:

- Authentication performed by the Integrated Solutions Console hosting the Tivoli Dynamic Workload Broker Web Console
- Authentication performed on the Tivoli Dynamic Workload Broker server side

> **Note:** Before reading the following sections make sure that you have read "Direct client Æ Server authentication" on page 60. We explained several terms in that section and they are necessary for understanding the following sections.

Now we describe the Integrated Solutions Console authentication mechanism in greater detail and explain the interaction with Tivoli Dynamic Workload Broker server authentication.

The Tivoli Dynamic Workload Broker Web Console is hosted by the Integrated Solutions Console. It enforces the authentication mechanism by default, has its own authentication mechanism, and maintains its own credential vault. The Integrated Solutions Console keeps all the defined credentials in the embedded Cloudscape™ database. The Integrated Solutions Console does not use any operating system authentication, nor does it integrate with any external LDAP. Each user that wants to access any application running within the Integrated Solutions Console (such as Tivoli Dynamic Workload Broker Web Console) is authenticated against the Integrated Solutions Console credential vault.

The user that wants to use the Tivoli Dynamic Workload Broker Web Console to access the Tivoli Dynamic Workload Broker server sees only one visible authentication step — he must provide the user ID and password defined in the Integrated Solutions Console.

If the Global Security on the WebSphere Application Server hosting the Tivoli Dynamic Workload Broker server is turned off, no further authentication is required.

If the Global Security on the WebSphere Application Server hosting the Tivoli Dynamic Workload Broker server is turned on, the user must authenticate itself against the WebSphere Application Server hosting the Tivoli Dynamic Workload Broker server. The logon credentials are not the same on the Integrated Solutions Console and on the operating system (or LDAP) that uses the WebSphere Application Server (hosting the Tivoli Dynamic Workload Broker server). Both credential vaults are *not automatically synchronized* in any way. Due to this, a mapping among the logon definitions stored in the Integrated Solutions Console and authority that is used by WebSphere Application Server (hosting the Tivoli Dynamic Workload Broker server) must be done. The mapping is performed on the Integrated Solutions Console side.

The mapped credentials are then passed to the WebSphere Application Server (hosting the Tivoli Dynamic Workload Broker server). Then the authentication mechanism is exactly the same, as described in “Direct client Æ Server authentication” on page 60. The credentials are transferred in each HTTP request within the HTTP header and the user’s identity is checked using HTTP Basic authentication.

Figure 2-13 shows the whole mechanism of authentication when the user connects to the Tivoli Dynamic Workload Broker server using the Web Console.

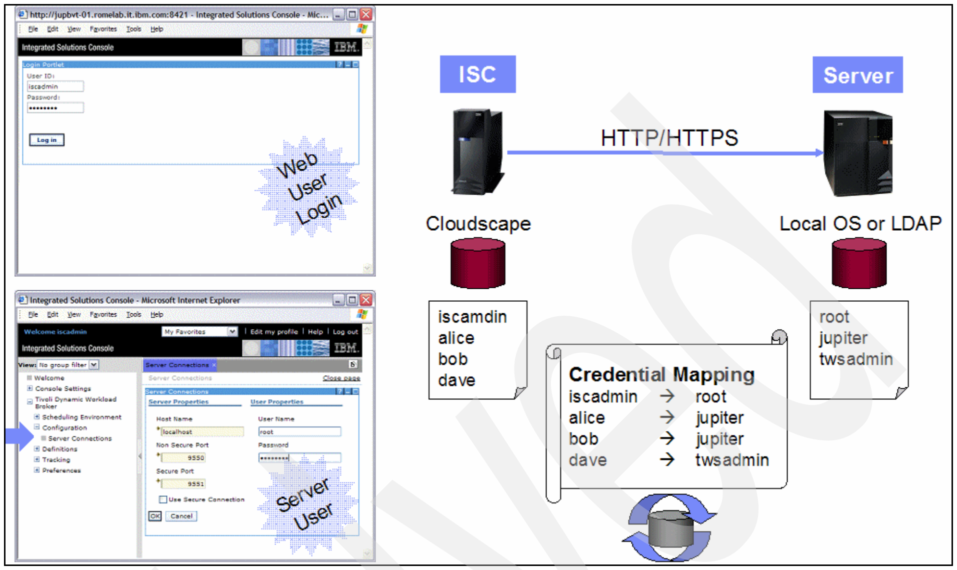


Figure 2-13 Browser-ISC-TDWB server authentication

2.7.4 Authorization roles

In this section we describe the different authorization roles that different Tivoli Dynamic Workload Broker users can posses. We also explain the scope of roles and where can they be defined.

Different degrees of authorization allow users to perform different set of actions. Table 2-2 shows the full list of all possible authorization roles.

Table 2-2 List of Tivoli Dynamic Workload Broker authorization roles

Authorization role	**Permitted operations**
Administrator	Super user with full authorization (configurator + developer + operator)
Configurator	Manages the scheduling infrastructure
Developer	Manages job definitions (necessary role for Job Brokering Definition Console when communication with Tivoli Dynamic Workload Broker server)

Authorization role	Permitted operations
Operator	Monitors and controls any job that has been submitted
Submitter	Monitors and controls own jobs (necessary role required for TWS Agent)

The user assignments to particular roles can be changed using the WebSphere Application Server Administrative Console (on the WebSphere Application Server hosting the Tivoli Dynamic Workload Broker server).

By default any user that successfully authenticates against the WebSphere Application Server (hosting the Tivoli Dynamic Workload Broker server) has the administrator role.

If this user accesses the Tivoli Dynamic Workload Broker server using the Web Console interface, he must authenticate itself at least against the Integrated Solutions Console.

Note: If a particular role is assigned to a user, it is valid for all jobs and resources in the environment. For instance, a developer role allows the user to modify and delete *any* job defined in the Job Repository. It is not possible to narrow the scope of jobs and resources to the given role. However, this feature is planned for future versions.

Recommended security best practices after installation

In this section we describe the necessary steps that will set up a more secure environment than the default installation:

1. Turn on WebSphere Application Server Global Security (on the WebSphere Application Server hosting the Tivoli Dynamic Workload Broker server).
2. Define new users within the Integrated Solutions Console.
3. Map these newly defined users with the users used by the WebSphere Application Server hosting the Tivoli Dynamic Workload Broker server.
4. Assign the users their desired authorization roles using the WebSphere Application Server Administrative Console (on the WebSphere Application Server hosting the Tivoli Dynamic Workload Broker server).
5. Optionally, generate a new server certificate for the Tivoli Dynamic Workload Broker server. This certificate will be used for communication in the client network. Distribute this new certificate into the appropriate directory on all possible clients. If you install any new client (such as a new instance of the

Job Brokering Definition Console, you must distribute this certificate to that client).

2.8 Tivoli Dynamic Workload Broker deployment scenarios

In this section we look at the different Tivoli Dynamic Workload Broker deployment scenarios covering not only the different ways that the Tivoli Dynamic Workload Broker can be deployed ether in a standalone solution or the integration with an enterprise scheduling tool, but also best practices around where the main components of the Tivoli Dynamic Workload Broker can be installed.

2.8.1 Location of main Tivoli Dynamic Workload Broker components

In this section we cover where the main components of a Tivoli Dynamic Workload Broker that can be installed, and the location of where these main components are installed depends on the customer, as they may already have some of these components installed and would like to use them for Tivoli Dynamic Workload Broker, and these are DB2 Universal database, WebSphere Application Server, Tivoli Agent Manager, the Integrated Solutions Console, and the Common Agent Services. For detailed information about the installation of each component look Chapter 3, “Tivoli Dynamic Workload Broker installation” on page 81, or the *IBM Tivoli Dynamic Workload Broker Installation and Configuration,* SC32-2282.

The main components of Tivoli Dynamic Workload Broker that are covered in this section are:

- DB2 Universal Database™ server or client
- WebSphere Application Server
- Tivoli Dynamic Workload Broker server
- Integrated Solutions Console
- Tivoli Dynamic Workload Broker Web Console
- Tivoli Dynamic Workload Broker Job Brokering Definition Console
- Tivoli Agent Manager
- Tivoli Common Agents
- Tivoli Dynamic Workload Broker agent

2.8.2 DB2 Universal Database

The DB2 Universal Database is a prerequisite for two of the other components, namely the Tivoli Dynamic Workload Broker server and Tivoli Agent Manager. You can use the same DB2 instance for both of these products or you can use a different DB2 instance for the the Tivoli Dynamic Workload Broker server and Tivoli Agent Manager. This depends on the performance of the database and the location of the Tivoli Dynamic Workload Broker server and Tivoli Agent Manager.

The location of the DB2 instance depends on the customer, as they may already have an instance of DB2 installed in their environment that meets the required software supported level, in which case you can use this for both the Tivoli Dynamic Workload Broker server and Tivoli Agent Manager. If this DB2 instance resides on a different system from that of the Tivoli Dynamic Workload Broker server and or the Tivoli Agent Manager then you will have to install the DB2 client for the Tivoli Dynamic Workload Broker server and the Tivoli Agent Manager.

For better performance it would be preferable to have the DB2 on a different system from that of the Tivoli Dynamic Workload Broker server and Tivoli Agent Manager. But if the Tivoli Dynamic Workload Broker server is configured in a small environment and running on a powerful system, then you can install DB2 on the same system as the Tivoli Dynamic Workload Broker server.

2.8.3 WebSphere Application Server

In a normal installation you would have one instance of WebSphere Application Server for both the Tivoli Dynamic Workload Broker server and the Tivoli Agent Manager. This WebSphere Application Servers applications can coexist on the same system as the server and Agent Manager, as the installation default ports are different for each instance.

For the best performance, we recommend installing the Tivoli Dynamic Workload Broker server and the Tivoli Agent Manager each into a separate instance of WebSphere Application Server, running on two different servers. But this depends on the customer, as they may have a WebSphere Application Server that meets the required support level that they would like to use for part of this installation.

2.8.4 Tivoli Dynamic Workload Broker server

The The Tivoli Dynamic Workload Broker server is an application that runs in the WebSphere Application server. So the installation location of the Tivoli Dynamic Workload Broker server depends on the installed location of the WebSphere Application server, as described above.

2.8.5 Tivoli Dynamic Workload Broker Web Console

The Tivoli Dynamic Workload Broker Web Console is an application that runs in the Integrated Solution Console. So the installation location of the Tivoli Dynamic Workload Broker Web console depends on the installed location of the Integrated Solution Console.

2.8.6 Tivoli Dynamic Workload Job Brokering Definition console

The Tivoli Dynamic Workload Broker Job Brokering Definition Console can be installed on any workstation that the application developer for the Tivoli Dynamic Workload Broker, this workstation requires an TCP/IP connection to the Tivoli Dynamic Workload Broker server, and providing that it meets the required support level for both hardware and operating system for the Tivoli Dynamic Workload Broker Job Brokering Definition Console product.

2.8.7 Tivoli Agent Manager

The location of where you would install the Tivoli Agent Manager depends on the Location of both DB2 and the WebSphere Application server, as the Tivoli Agent Manager has a prerequisite of both of these products. The Tivoli Agent Manager may already be installed in your environment, as Tivoli Provisioning Manager also uses the Tivoli Agent Manager. If this is the case then you can use this installation for the Tivoli Dynamic Workload Broker.

We recommend that the Tivoli Agent Manager is installed on a separate system from the the Tivoli Dynamic Workload Broker server so that the load of these two products is spread across two systems. You can install the Tivoli Agent Manager and Tivoli Dynamic Workload Broker on the same system, as the default ports are different for each of these two products. You would only install both of these products on the same system if the Dynamic Workload Broker server is installed in a small environment and is running on a powerful system that is able to handle the load.

2.8.8 Tivoli Common Agents

The Common Agents are the agents hosting the Tivoli Dynamic Workload Broker subagent and the Tivoli Common Inventory Technology subagent providing them with remote deployment capability, management, and security. These systems must meet the required support level for both hardware and operating system, as listed in *IBM Tivoli Dynamic Workload Broker Installation and Configuration,* SC32-2282.

2.8.9 Tivoli Dynamic Workload Broker agent

The Tivoli Dynamic Workload Broker agent is where the jobs will be executed, so you will need to install an instance of the Tivoli Dynamic Workload Broker agent on each system on which you require jobs to be run. These systems must meet the required support level for both hardware and operating system, as listed in *IBM Tivoli Dynamic Workload Broker Installation and Configuration,* SC32-2282.

2.8.10 Tivoli Dynamic Workload Broker standalone solution

The Tivoli Dynamic Workload Broker can be installed as a standalone solution without the integration of an enterprise scheduling tool like the Tivoli Workload Scheduler. You would choose this type of configuration for a number of reasons:

- Tivoli Dynamic Workload Broker may be installed in a very small environment where all the jobs can be defined and controlled by the two consoles, namely the Job Brokering Definition console, which is used for creating and modifying the jobs; and the Tivoli Dynamic Workload Broker Web Console, which is used for managing the Tivoli Dynamic Workload Broker environment.
- If there are no complex dependences between jobs you have the ability to create the jobs and submit them through the two consoles.
- If the Tivoli Workload Scheduler is not installed in this environment.

Figure 2-14 shows the schema of the Tivoli Dynamic Workload Broker standalone solution.

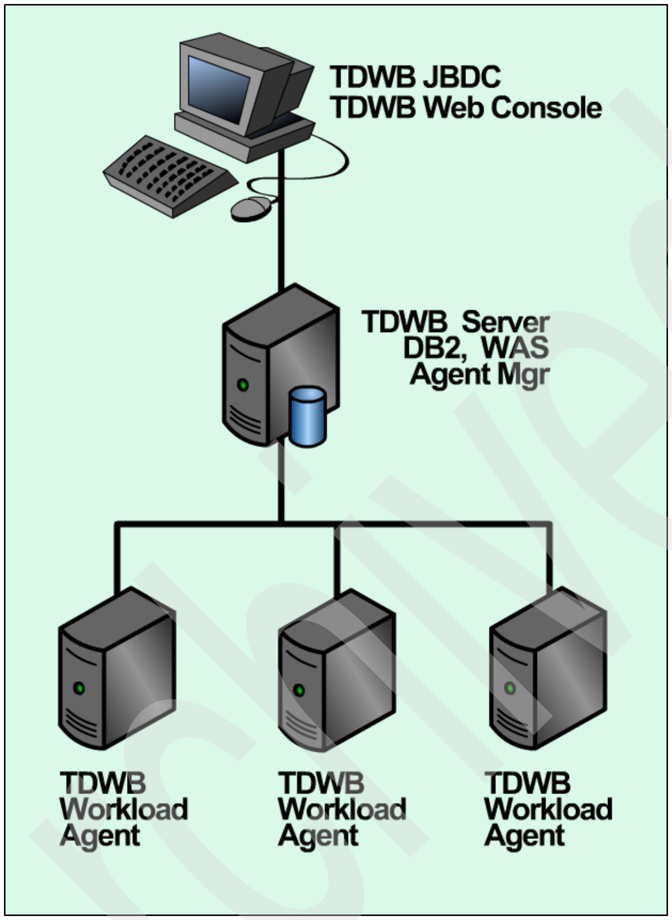


Figure 2-14 Standalone Tivoli Dynamic Workload Broker solution

2.8.11 Common usage of Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker

A more common solution is where we integrate Tivoli Dynamic Workload Broker with Tivoli Workload Scheduler, as Tivoli Dynamic Workload Broker provides a series of improvements on your existing Tivoli Workload Scheduler solution:

- Virtualization of the scheduling infrastructure by providing an abstraction layer on the resource selection

- Workload balancing by routing jobs among a group of resources according to the availability and activity levels of those resources
- SOA job brokering services
- Scheduling of IBM WebSphere Java 2 Enterprise Edition (J2EE) applications
- Automatic routing of jobs to the most appropriate resources based on job requirements
- Enhanced flexibility in workload distribution and running
- Automatic routing of jobs for which submission failed to appropriate resources

Tivoli Workload Scheduler provides the following features to Tivoli Dynamic Workload Broker not only in a distributed environment but also in a end-to-end deployment as well:

- End-to-end scheduling infrastructure
- Ability to use all of the sophisticated scheduling dependices such as:
  - Planning where we can define run cycles using calendars or specifying days of week, using work days
  - Defining timed dependency both in start and deadline time
  - Creation of a file
  - Prompts
  - Submission priority
  - Ability to define the number of simultaneously executing jobs
  - Choreographing capabilities where we have the ability to define predecessors and successors

Figure 2-15 shows the schema of the Tivoli Dynamic Workload Broker integrated with the Tivoli Workload Scheduler.

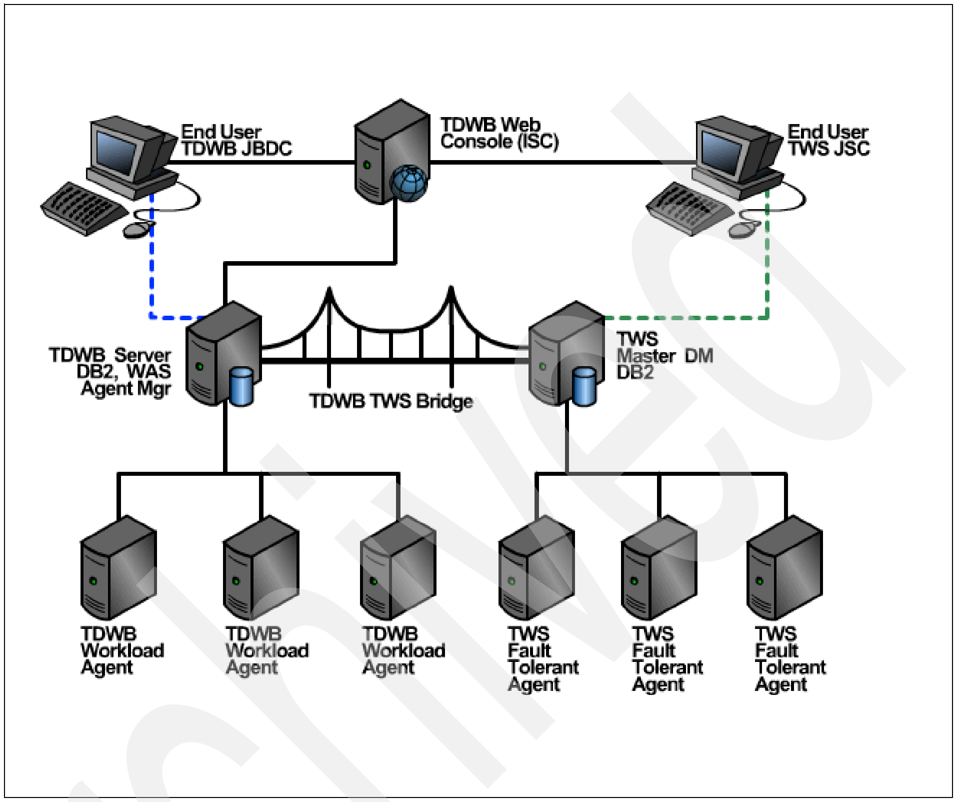


Figure 2-15 Tivoli Dynamic Workload Broker integrated with Tivoli Workload Scheduler

2.8.12 Setting up monitoring for Tivoli Dynamic Workload Broker

Independently on the selected deployment scenario, it is always good to integrate the job brokering environment with the monitoring solution. The operators of monitoring centers usually observe the monitoring consoles more often and can quickly react to errors or unpredictable conditions.

An alert is usually set up together with monitoring. Due to this you can receive (for instance) an SMS on your mobile device each time an important error occurs in the Tivoli Dynamic Workload Broker environment.

A more detailed description of integration of the Tivoli Dynamic Workload Broker with IBM monitoring solution - IBM Tivoli Monitoring is in 8.4, “Integration with IBM Tivoli Monitoring” on page 325.

2.9 Physical location of Tivoli Dynamic Workload Broker's components

This section provides a more detailed view of Tivoli Dynamic Workload Broker components from the file system point of view. It can serve as a basic link from the architectural point of view to the configuration or troubleshooting perspective.

Note: Be aware that the values provided below are default paths and may vary depending on your installation. They serve as a basic pointer only.

2.9.1 Locations of server components

As described before, Tivoli Dynamic Workload Broker server components consist of several J2EE applications:

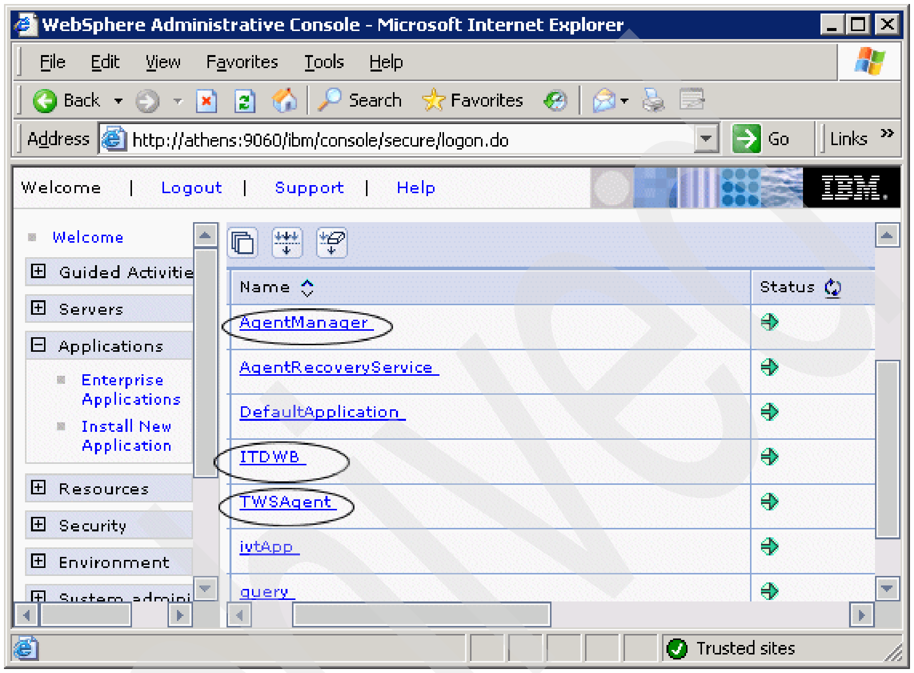
- Main application called ITDWB
- Agent Manager
- Optional Tivoli Workload Scheduler Agent

All of these enterprise applications are running within the WebSphere Application Server. Each of these components can be managed separately by the WebSphere Administrative Console and has its own file structure within the WebSphere Application Server.

The Tivoli Dynamic Workload Broker Web Console is not a direct part of Tivoli Dynamic Workload Broker server binaries. It is installed as a part of the *Integrated Solutions Console*. Integrated Solutions Console is a portal solution that can be a common interface for other products, such as Tivoli Storage Manager (TSM), Tivoli Workload Scheduler (TWS), and so on. The Integrated Solutions Console uses its own embedded WebSphere Application Server instance.

Figure 2-16 on page 74 shows the Tivoli Dynamic Workload Broker server components as separate J2EE applications that are running within the WebSphere Application Server. Note that neither Tivoli Dynamic Workload Broker Web Console nor Integrated Solutions is present within the list. The reason for this is that the Tivoli Dynamic Workload Broker Web Console and Integrated Solutions Console are installed under other WebSphere Application

Server instances. In the current release the Integrated Solutions Console must use its own embedded WebSphere Application Server, and this cannot be shared with the ITDWB enterprise application (Tivoli Dynamic Workload Broker server).


WebSphere Administrative Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://athens:9060/ibm/console/secure/logon.do Go Links
Welcome | Logout | Support | Help
Welcome
Guided Activitie
Servers
Applications
Enterprise Applications
Install New Application
Resources
Security
Environment
System admini
Trusted sites

Name	Status
AgentManager	➔
AgentRecoveryService	➔
DefaultApplication	➔
ITDWB	➔
TWSAgent	➔
ivtApp	➔
query	➔

Figure 2-16 List of server components (WebSphere Management Console)

Default locations of server components

Table 2-3 and Table 2-4 on page 76 show the default paths for server components on Windows, UNIX, and Linux platforms.

Important: In the tables below we point to directories for WebSphere Application Server enterprise applications. We show these directories because we want to stress what server components are independent enterprise applications. We do not recommend modifying any file within those directory trees, nor to invoke any command from them. Doing so could cause your Tivoli Dynamic Workload Broker server to become unstable.

Table 2-3 Default server paths on Windows platform

Component	**Path**
WebSphere installation directory	C:\Program Files\IBM\WebSphere\AppServer
Tivoli Dynamic Workload Broker components directories (WebSphere enterprise applications)	C:\Program Files\IBM\WebSphere\AppServer\profiles\default\installedApps\<*node_cell_name*>\ITDWB.ear
Tivoli Dynamic Workload Broker installation directory (command line binaries and others)	C:\Program Files\IBM\ITDWB\Server
Tivoli Dynamic Workload Broker TWS Agent	C:\Program Files\IBM\WebSphere\AppServer\profiles\default\installedApps\<node_cell_name>\TWSAgent.ear
ISC WebSphere installation directory	C:\Program Files\IBM\ISC\AppServer
ISC enterprise application directory	C:\Program Files\IBM\ISC\AppServer\profiles\default\installedApps\DefaultNode

Note: The value of *<node_cell_name>* used in Table 2-3 on page 75 typically consists of machine's host name keywords Node, Cell, and a number. In our scenario, the default generated value for machine ATHENS was athensNode01Cell.

Table 2-4 Default server paths on UNIX and Linux platforms

Component	Path
WebSphere installation directory	/usr/IBM/WebSphere/AppServer (AIX) /opt/IBM/WebSphere/AppServer (other UNIXes and Linux)
Tivoli Dynamic Workload Broker components directories (Websphere enterprise applications)	/usr/IBM/WebSphere/AppServer/ profiles/default/ installedApps/ `<node_cell_name>`/ITDWB.ear (AIX) /opt/IBM/WebSphere/AppServer/ profiles/default/ installedApps/ `<node_cell_name>`/ITDWB.ear (other UNIXes and Linux)
Tivoli Dynamic Workload Broker installation directory (command-line binaries and others)	/opt/IBM/ITDWB/Server
Tivoli Dynamic Workload Broker TWS Agent	/usr/IBM/WebSphere/AppServer/ profiles/default/ installedApps/ `<node_cell_name>`/TWSAgent.ear (AIX) /opt/IBM/WebSphere/AppServer/ profiles/default/ installedApps/`<node_cell_name>`/ TWSAgent.ear (other UNIXes and Linux)
ISC WebSphere installation directory	/opt/IBM/ISC/AppServer
ISC enterprise application directory	/usr/IBM/ISC/AppServer/IBM/ISC/ AppServer/profiles/default/ installedApps/DefaultNode (AIX) /opt/IBM/ISC/AppServer/IBM/ISC/ AppServer/profiles/default/ installedApps/DefaultNode (other UNIXes and Linux)

2.9.2 Locations of agent components

Table 2-5 and Table 2-6 show the default paths for agent components on Windows and UNIX platforms.

Table 2-5 Default agent paths on Windows platform

Component	Path
Common Agent (if installed with Tivoli Dynamic Workload Broker agent)	C:\Program Files\IBM\ITDWB\Agent
Tivoli Dynamic Workload Broker agent	C:\Program Files\IBM\ITDWB\Agent\ep\runtime\agent\subagents

Table 2-6 Default agent paths on UNIX and Linux platforms

Component	Path
Common Agent (if installed with Tivoli Dynamic Workload Broker Agent)	/opt/IBM/ITDWB/Agent
Tivoli Dynamic Workload Broker agent	/opt/IBM/ITDWB/Agent/ep/runtime/agent/subagents

2.9.3 Location of certificates and private keys

In this section we list the default keystore locations. We provide two tables, listing the keystores used in a client network and agent network, as described in "Two communication networks" on page 53.

Keystores for client network

Table 2-7 shows the location of the keystores of private keys and certificates necessary on a client network for server-side SSL handshake. The keystores are by default located on file systems as described in the table. They are provided out-of-box and are not generated at installation time.

Table 2-7 Keystores for client network

Security file type	**Path and file name**
Job Brokering Definition Console - server's certificate	*<JBDC_install_dir>*/Certs/ TDWBClientTrustFile.jks
Job Brokering Definition Console - client's private key	*<JBDC_install_dir>*/Certs/ TDWBClientKeyFile.jks
Web Console installed within Integrated Solutions Console - server's certificate	*<ISC_WebSphere_install_dir>*/ISC/ AppServer/profiles/default/etc/ TDWBClientTrustFile.jks
Web Console installed within Integrated Solutions Console - client's private key	*<ISC_WebSphere_install_dir>*/ISC/ AppServer/profiles/default/etc/ TDWBClientKeyFile.jks
Command-line interface installed on server - servers's certificate	*<ITDWB_server_install_dir>*/Certs/ TDWBClientTrustFile.jks
Command-line interface installed on server - client's private key	*<ITDWB_server_install_dir>*/Certs/ TDWBClientKeyFile.jks
Clients' certificates stored on server	*<ITDWB_server_install_dir>*/Certs/ TDWBServerTrustFile.jks
Server's private key	*<ITDWB_server_install_dir>*/Certs/ TDWBServerKeyFile.jks

Keystores for agent network

Table 2-8 shows the location of the keystores of private keys and certificates necessary on agent network for a mutual SSL handshake. The keystores are by default located on file systems, as described in the table. Private keys and certificates are generated by Agent Manager's Certification Authority in the registration time.

Table 2-8 Keystores fo agent network

Security file type	**Path and file name**
Certificate of Agent Manager's Certification Authority transferred to Tivoli Dynamic Workload Broker server at start of the registration.	*<ITDWB_server_install_dir>*/ ResourceManager\cert\agentTrust.jks
Certificate and private key issued by Agent Manager's Certification Authority for Tivoli Dynamic Workload Broker server.	*<ITDWB_server_install_dir>*/ ResourceManager\cert\agentKeys.jks
Certificate of Agent Manager's Certification Authority transferred to Common Agent at start of the registration.	*<common_agent_install_dir>*/ep/runtime/agent/cert/agentTrust.jks
Certificate and private key issued by Agent Manager's Certification Authority for Common Agent.	*<common_agent_install_dir>*/ep/runtime/agent/cert/agentKeys.jks
Certificate of Agent Manager.	*<ITDWB_server_install_dir>*/ AgentManager/certs/ agentManagerTrust.jks
Private key of Agent Manager.	*<ITDWB_server_install_dir>*/ AgentManager/certs/ agentManagerKeys.jks
Certificates of Agent Manager's Certification Authority transferred from Agent Manager to Common Agent or Tivoli Dynamic Workload Broker server at start of registration. This certificate is used by agent (resource manager) for initiating server-side SSL handshake.	*<ITDWB_server_install_dir>*/ AgentManager/certs/agentTrust.jks
Certificate and private key of Agent Manager's Certification authority.	*<ITDWB_server_install_dir>*/ AgentManager/certs/CARootKeyRing.jks
Certificate Revocations List of Agent Manager's Certification Authority.	*<ITDWB_server_install_dir>*/ AgentManager/certs/ CertificateRevocationList

3

Tivoli Dynamic Workload Broker installation

In this chapter we provide high-level technical information about the required software needed before the installation of the Tivoli Dynamic Workload Broker components. This chapter also contains the installation steps of typical and custom setups.

In this chapter the following topics are discussed:

3.1 Introduction

For a completely functional Tivoli Dynamic Workload Broker scheduling network integrated with Tivoli Workload Scheduler there are nine Tivoli Dynamic Workload Broker components that need to be installed. Some of the components will be installed by default if the typical installation is performed and some components can be installed on separate machines. The purpose of this installation documentation is to present the components to install, the prerequisites, the order of install, and a brief explanation of how the components interrelate so that install planning can be accomplished quickly.

This is a list of the Tivoli Dynamic Workload Broker components listed in the order of when they should be installed:

1. Tivoli Dynamic Workload Broker server
2. Tivoli Dynamic Workload Broker Web Console
3. Tivoli Dynamic Workload Broker Job Brokering Definition Console
4. Tivoli Dynamic Workload Broker Broker agent

This is a list of the Tivoli Dynamic Workload Broker components and their sub components. The list shows what order components will be installed in if the typical installation is performed, and this is the order that will need to be performed if performing the install using a custom install.

1. Tivoli Dynamic Workload Broker server
   - Tivoli Agent Manager database
   - Tivoli Agent Manager
   - Tivoli Dynamic Workload Broker server database
   - Tivoli Dynamic Workload Broker server
   - Tivoli Workload Scheduler agent
2. Tivoli Dynamic Workload Broker Web Console
   - Integrated Solutions Console (ISC)
   - Tivoli Dynamic Workload Broker Web Console
3. Tivoli Dynamic Workload Broker Job Brokering Definition Console
4. Tivoli Dynamic Workload Broker agent

This is the list of Tivoli Dynamic Workload Broker components with sub components and software prerequisites:

- Tivoli Dynamic Workload Broker server
  - Tivoli Agent Manager database
    - Prerequisite: DB2 (Server or Client)
  - Tivoli Agent Manager
    - Prerequisite: WebSphere Application Server 6.0.2.11

- Tivoli Dynamic Workload Broker server database
  - Prerequisite: DB2 (Can be the same as the one used for Tivoli Agent Manager database)

    Note: Oracle support for the Tivoli Dynamic Workload Broker server will be available in Tivoli Dynamic Workload Broker V1.2.

- Tivoli Dynamic Workload Broker server
  - Prerequisite: WebSphere Application Server 6.0.2.11.

    Note: If the Tivoli Dynamic Workload Broker server is installed on the same host as the Tivoli Agent Manager then the same WebSphere Application server will be used.

  - Prerequisite: Tivoli Agent Manager
- Tivoli Workload Scheduler agent
  - Prerequisite: Tivoli Dynamic Workload Broker server

- Tivoli Dynamic Workload Broker Web Console
  - Integrated Solutions Console (ISC)
  - Tivoli Dynamic Workload Broker Web Console
    - Prerequisite/Corequisite: Integrated Solutions Console (ISC)
- Tivoli Dynamic Workload Broker Job Brokering Definition Console
- Tivoli Dynamic Workload Broker agent

3.2 Planning for installation

Prior to approaching the Tivoli Dynamic Workload Broker component installation there are a couple of components that have specific software requirements. DB2 UDB must be installed prior to installing the Tivoli Agent Manager Database and the Tivoli Dynamic Workload Broker server database. WebSphere Application Server V 6.0.2.11 must be installed so that Tivoli Agent Manager and the Tivoli Dynamic Workload Broker server can be installed.

3.2.1 Tivoli Dynamic Workload Broker software prerequisites

In this section we discuss the Tivoli Dynamic Workload Broker software prerequisites.

DB2

The following components must have an existing DB2 Server or client installed:

- Tivoli Agent Manager database
- Tivoli Dynamic Workload Broker server database

DB2 Versions

The versions are:

- DB2 Universal Database (UDB) Version 8.2 client or server with Fix Pack 1 or later
- DB2 Universal Database Version 8.1 client or server with Fix Pack 8 or later

Note: DB2 UDB Version 8.1 Fixpack 7 is equivalent to DB2 UDB Version 8.2 GA. DB2 UDB Version 8.1 Fixpack 8 is also known as Version 8.2 Fixpack 1. Version 8.1 Fixpack 9 is also known as Version 8.2 Fixpack 2, and so on.

Installing DB2 server or client considerations

When your required level of reliability, availability, and scalability are met by a locally installed database on the same machine as the Tivoli Dynamic Workload Broker server or even when dictated by a remote database, which might even be partitioned across a number of nodes in a cluster, DB2 Universal Database meets these requirements.

To obtain the system prerequisites for DB2 (UDB) installations for operating systems that support the Tivoli Dynamic Workload Broker server, see:

- AIX:

  http://www-1.ibm.com/support/docview.wss?rs=71&uid=swg21181544

  Note: Known issues for DB2 on AIX 4.3.3, 5.1, 5.2, and 5.3 are discussed at:

  http://www-1.ibm.com/support/docview.wss?rs=71&uid=swg21165448

- Linux:

  http://www-306.ibm.com/software/data/db2/linux/validate/platdist82.html

- Windows:

  http://www-1.ibm.com/support/docview.wss?rs=71&uid=swg21176759

For the most up-to-date operating system information see:

http://www.ibm.com/software/data/db2/udb/sysreqs.html

One early space consideration for DB2 is the install location for the server or client. The location on UNIX systems cannot be modified:

- AIX: /usr/opt/db2_08_01
- Linux: /opt/IBM/db2/V8.1

WebSphere Application Server

WebSphere Application Server (WAS) is the home to both the Tivoli Agent Manager and the Tivoli Dynamic Workload Broker server applications. These applications are recommended to be installed to a single WAS on the same machine.

Note: Tivoli Agent Manager is installed along with the Tivoli Dynamic Workload Broker server by default (using the typical installation). If the Tivoli Agent Manager and the Tivoli Dynamic Workload Broker server are installed on separate hosts, then consider the Tivoli Agent Manager as a prerequisite of the Tivoli Dynamic Workload Broker server.

- WebSphere Application Server version

  WebSphere Application Server, Version 6.0, 32-bit version, with Refresh Pack 2 and Fix Pack 11.

- WebSphere patch image locations
  - 6.0.2: WebSphere Application Server V6.0 Refresh Pack 2:

    `http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg24010066`
  - 6.0.2.11: WebSphere Application Server V6.0.2 Fix Pack 11:

    `http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg24012484`

WebSphere patch install instructions

Refer to the readme documents for specific installation instructions:

- 6.0.2: WebSphere Application Server V6.0 Refresh Pack 2:

  `http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg27006336`
- 6.0.2.11: WebSphere Application Server V6.0.2 Fix Pack 11:

  `http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg27007828`

Important steps for each fixpack installation are found in the readme files. For the instructions for UNIX include:

1. Run the **`backupConfig`** command to back up configuration files. See “Backing up and restoring administrative configurations” in the online information center.

2. Back up and remove the older updateinstaller directory. For example:
   a. `cd /usr/IBM/WebSphere/AppServer`
   b. `tar -cvf updateinstaller_old.tar updateinstaller`

      Back the updateinstaller directory recursively using the tar create feature.
   c. `rm -r updateinstaller`
3. Extract the fixpack tar image in the install_root directory. For example:
   a. `cd /usr/IBM/WebSphere/AppServer`
   b. `# tar -xvf  /tmp/downloads/6.0-WS-WAS-AixPPC32-RP0000002.tar`

      The Update Installer for WebSphere Software is in the install_root/updateinstaller directory. The maintenance package is in the install_root/updateinstaller/maintenance directory.

The installation of the WebSphere Application Server is a prerequisite for all the applications requiring an application server:

- Tivoli Dynamic Workload Broker server
- Tivoli Agent Manager

WebSphere Application Server installation considerations

WebSphere Application Server V6.0.2 hardware requirements summary can be found at:

http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg27007250

WebSphere Application Server V6.0.2 detailed system requirements can be found at:

http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg27007256

Note: Tivoli Dynamic Workload Broker running on 64 bit computers is supported only if the IBM WebSphere Application Server Java Virtual Machine is the 32-bit version

3.2.2 Tivoli Dynamic Workload Broker hardware prerequisites

In this section we discuss the hardware prerequisites for the Tivoli Dynamic Workload Broker.

Supported operating systems

Table 3-2 on page 89 shows the supported operating system for each main component or prerequisite of Tivoli Dynamic Workload Broker.

Table 3-1 Supported operating systems

	Tivoli Dynamic Workload Broker server	Tivoli Dynamic Workload Broker agents (endpoints)	Tivoli Dynamic Workload Broker Job Brokering Definition Console	Tivoli Dynamic Workload Broker Web Console
AIX				
AIX 5.2	Yes	Yes	No	Yes
AIX 5.3	Yes	Yes	No	Yes
Linux				
SuSe Linux Enterprise Server (SLES) 8/UnitedLinux (UL) 1.0 for IA32	Yes	Yes	Yes	Yes
SuSe Linux Enterprise Server (SLES) 8/UL 1.0 for s/390 and zSeries®	Yes	Yes	No	Yes
SLES 8/UL 1.0 for pSeries®, 64-bit	Yes	Yes	No	Yes
SLES 8/UL 1.0 for iSeries®	No	No	No	Yes
SLES 9 for IA32	Yes	Yes	Yes	Yes
SLES 9 for s/390 and zSeries	Yes	Yes	No	Yes
SLES 9 with SP1, for pSeries, 64-bit	Yes	Yes	No	Yes
SLES 9 for iSeries	No	No	No	Yes
RHEL AS+ 3.0 IA32	Yes	Yes	Yes	Yes
RHEL 3.0 for S/390® and zSeries	Yes	Yes	No	Yes
RHEL 3.0 for pSeries, 64-bit	Yes	Yes	No	Yes
RHEL 3.0 for iSeries	No	No	No	Yes

Table 3-1 Supported operating systems

	Tivoli Dynamic Workload Broker server	Tivoli Dynamic Workload Broker agents (endpoints)	Tivoli Dynamic Workload Broker Job Brokering Definition Console	Tivoli Dynamic Workload Broker Web Console
RHEL 4.0 for IA32	Yes	Yes	Yes	Yes
RHEL 4.0 - AMD64	No	Yes	No	No
RHEL 4.0 for s/390 and zSeries	Yes	Yes	No	Yes
RHEL 4.0 for pSeries, 64-bit	Yes	Yes	No	Yes
RHEL 4.0 for iSeries	No	No	No	Yes
Windows				
Windows 2000 Server	Yes	Yes	Yes	Yes
Windows 2000 Advanced Server	Yes	Yes	Yes	Yes
Windows 2000 Data Center Server	No	No	No	No
Windows XP Professional	No	No	Yes	Yes
Windows Server® 2003 Standard	Yes	Yes	Yes	Yes
Windows Server 2003 Enterprise	Yes	Yes	No	Yes
Windows Server 2003 Enterprise AMD64/EM64T	No	Yes	No	No

Notes:

1. Windows Hotfix 906868 is also a prerequisite when using any of the Windows Server 2003 systems.
2. The Tivoli Dynamic Workload Broker running on 64-bit computers is supported only if the IBM WebSphere Application Server Java Virtual Machine is the 32-bit version.
3. VMware configurations with supported operating systems will be supported. Consider that the Tivoli Dynamic Workload Broker component memory requirement.

For system requirements and latest information about Tivoli Dynamic Workload Broker, refer to the download document at the following link:

http://ibm.com/support/docview.wss?rs=3190&uid=swg24013539

Memory, disk space, and install authority prerequisites

Table 3-2 lists the memory, disk space, and install authority prerequisites for Tivoli Dynamic Workload Broker.

Table 3-2 Hardware and install authority prerequisites per component

	Tivoli Dynamic Workload Broker server	**Tivoli Dynamic Workload Broker agents (endpoints)**	**Tivoli Dynamic Workload Broker Job Brokering Definition Console**	**Tivoli Dynamic Workload Broker Web Console**
Memory	2 GB or more per processor 3 GB recommended.	512 MB or more per processor.	1 GB or more per processor.	1 GB or more per processor. 2 GB recommended.
Pre-install disk space	600 MB free.	250 MB free.	300 MB free.	600 MB free.
Installed disk space	500 MB.	150 MB.	200 MB.	500 MB.
Install authority	Log in as root (UNIX), administrator (Windows).	Log in as root (UNIX), administrator (Windows).	Log in as root (UNIX), administrator (Windows).	Log in as root (UNIX), administrator (Windows).

Note: The required amount of memory is for each component. If one installs more than a single component on a computer then the amount of memory specified for each component must be added together to obtain the total memory requirement.

3.2.3 Tivoli Dynamic Workload Broker network

The ideal Tivoli Dynamic Workload Broker network will have components installed on different hosts in an effort to find a balance between minimizing configuration complexity and maximizing performance. The following is an example.

Let us suppose that we have the following systems:

- Three AIX 5.3 machines: AIX1, AIX2, and AIX3
- One Windows 2003 Advance Server host: W2K1
- Two RHEL AS+ 3.0 IA32: RHEL1 and RHEL2
- Ten Linux SLES 9 for IA32 machines: SLES1, SLES2...SLES10

We can install the Tivoli Dynamic Workload Broker components and prerequisites as follows (see Figure 3-1 on page 91):

- DB2 Server on AIX1.
- Tivoli Dynamic Workload Broker server on AIX2. Install the DB2 Administrative Client and WebSphere Application Server as prerequisites. Point to the DB2 Server on AIX1 for database connections. Subsequently, the Tivoli Workload Scheduler Agent will be installed on this host.
- Tivoli Dynamic Workload Broker Web Console on RHEL1.
- Tivoli Dynamic Workload Broker Job Brokering Definition Console on W2K1. Also consider the W2K1 system as a local desktop.
- Tivoli Dynamic Workload Broker agents on AIX3, RHEL2, SLES1, SLES2...SLES10.

Note: Do not install a Tivoli Dynamic Workload Broker agent to a Tivoli Dynamic Workload Broker server host. Having an agent on the server could affect the resource availability calculations.

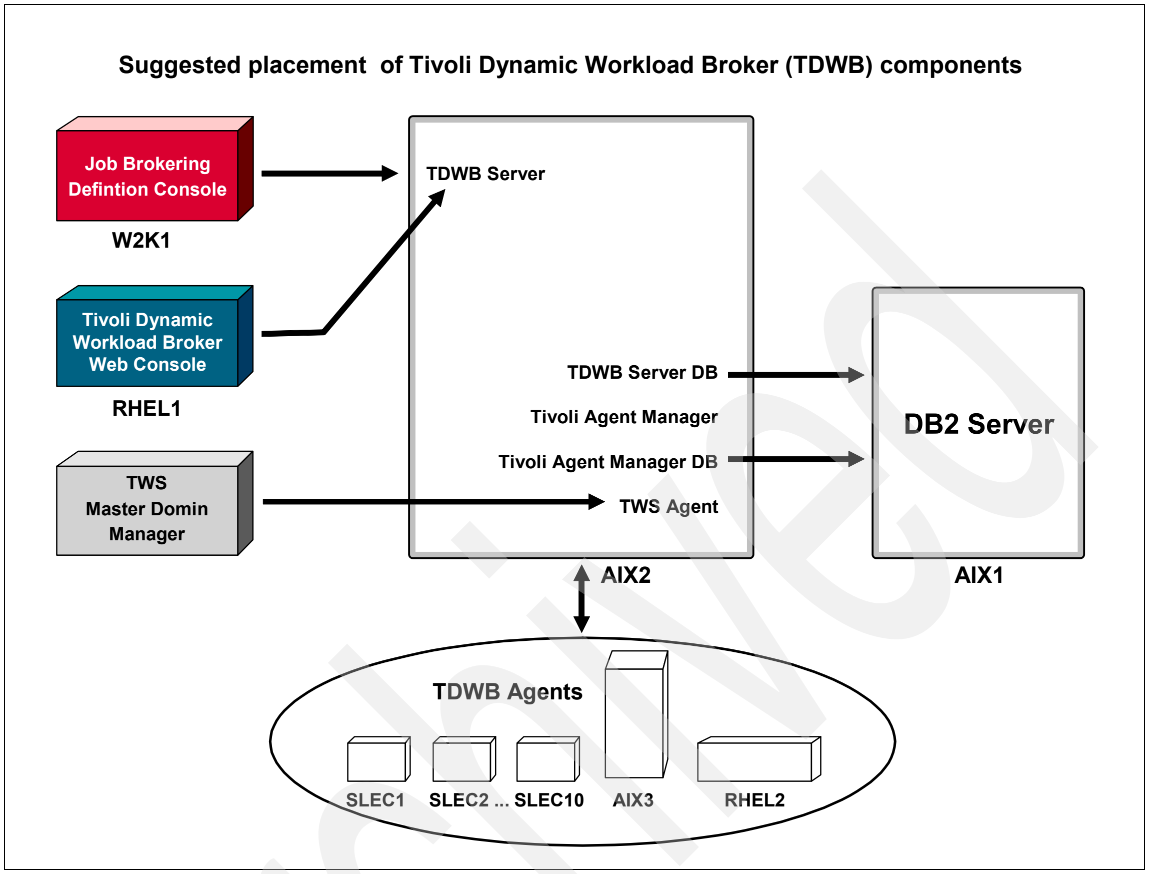


Figure 3-1 Sample placement of Tivoli Dynamic Workload Broker components

3.3 Installation

This section discusses the step-by-step installation of the Tivoli Dynamic Workload Broker. Refer to *IBM Tivoli Dynamic Workload Broker Installation and Configuration,* SC32-2282, for more information about these steps.

3.3.1 Choosing the installation method

The are two methods of performing an installation:

- Installing with the installation wizard

  Using this method, you can perform either a Typical installation or a Custom installation.

- Silent installation

  Using this method, you create a response file that contains all the installation parameters. You then run the installation from the command line using the response file.

There are two options for installing with the installation wizard:

- Typical install
- Custom install

If the installation fails, you can correct the error by resuming the installation. See 10.1.4, “Diagnose failure dialogue - using the step list” on page 493.

Default installation directories

Each component of the Tivoli Dynamic Workload Broker has its own default installation directory. You can use Table 3-3 as a reference in your installation. See also Table 2-3 on page 75, Table 2-4 on page 76, and Table 2-5 on page 77 for more details on the default installation locations.

Table 3-3 Default component installation locations

Tivoli Dynamic Workload Broker component	Default installation directory
Server	Windows: C:\Program Files\IBM\ITDWB\Server
	UNIX: /opt/IBM/ITDWB\Server
Tivoli Agent Manager	Windows: C:\Program Files\IBM\ITDWB\Server\AgentManager
	UNIX: /opt/IBM/ITDWB\Server\AgentManager
Web Console	Windows: C:\Program Files\IBM\ISC
	UNIX: /opt/IBM/ISC
Job Brokering Definition Console	Windows: C:\Program Files\IBM\ITDWB\JBDC
	UNIX: /opt/IBM/ITDWB\JBDC

Tivoli Dynamic Workload Broker component	Default installation directory
Agent	Windows: C:\Program Files\IBM\ITDWB\Agent
	UNIX: /opt/IBM/ITDWB\Agent

Locating the installation programs

The Tivoli Dynamic Workload Broker server installation programs are located in the root directory of the first CD.

There are seven CDs in the package for the Tivoli Dynamic Workload Broker. Each CD contains the launchpad, which is used to install each component. For example, you can initiate the launchpad from any of the seven CDs, and when you choose to install a component that is located on a different CD, you will be prompted to insert the correct one.

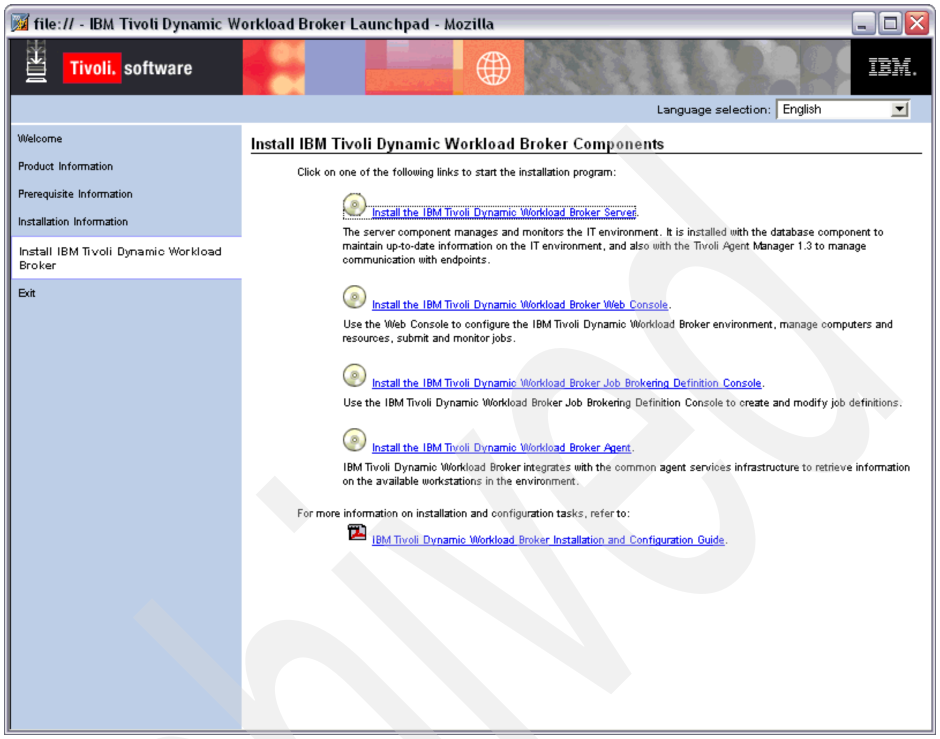
Starting the launchpad

The installation program uses a Java Virtual Machine that requires 10 MB of free space in the operating system's default temporary directory. You can start the install via the launchpad, or access each component's individual setup file.

The procedure for starting the launchpad installation program is:

1. With the correct CD in the drive, start the launchpad:
   - Windows - If the launchpad does not start automatically, run the launchpad.exe file from the root directory of the CD.
   - UNIX - Run the launchpad.sh file from the root directory of the CD.

2. The launchpad is displayed, as shown in Figure 3-2.


file:// - IBM Tivoli Dynamic Workload Broker Launchpad - Mozilla
Tivoli. software
IBM.
Language selection: English
Welcome
Product Information
Prerequisite Information
Installation Information
Install IBM Tivoli Dynamic Workload Broker
Exit

Install IBM Tivoli Dynamic Workload Broker Components

Click on one of the following links to start the installation program:

Install the IBM Tivoli Dynamic Workload Broker Server.

The server component manages and monitors the IT environment. It is installed with the database component to maintain up-to-date information on the IT environment, and also with the Tivoli Agent Manager 1.3 to manage communication with endpoints.

Install the IBM Tivoli Dynamic Workload Broker Web Console.

Use the Web Console to configure the IBM Tivoli Dynamic Workload Broker environment, manage computers and resources, submit and monitor jobs.

Install the IBM Tivoli Dynamic Workload Broker Job Brokering Definition Console.

Use the IBM Tivoli Dynamic Workload Broker Job Brokering Definition Console to create and modify job definitions.

Install the IBM Tivoli Dynamic Workload Broker Agent.

IBM Tivoli Dynamic Workload Broker integrates with the common agent services infrastructure to retrieve information on the available workstations in the environment.

For more information on installation and configuration tasks, refer to:

IBM Tivoli Dynamic Workload Broker Installation and Configuration Guide.

Figure 3-2 The installation launchpad

3. Click **Install IBM Tivoli Dynamic Workload Broker** on the left side of the screen.

   From this screen you have the following options:

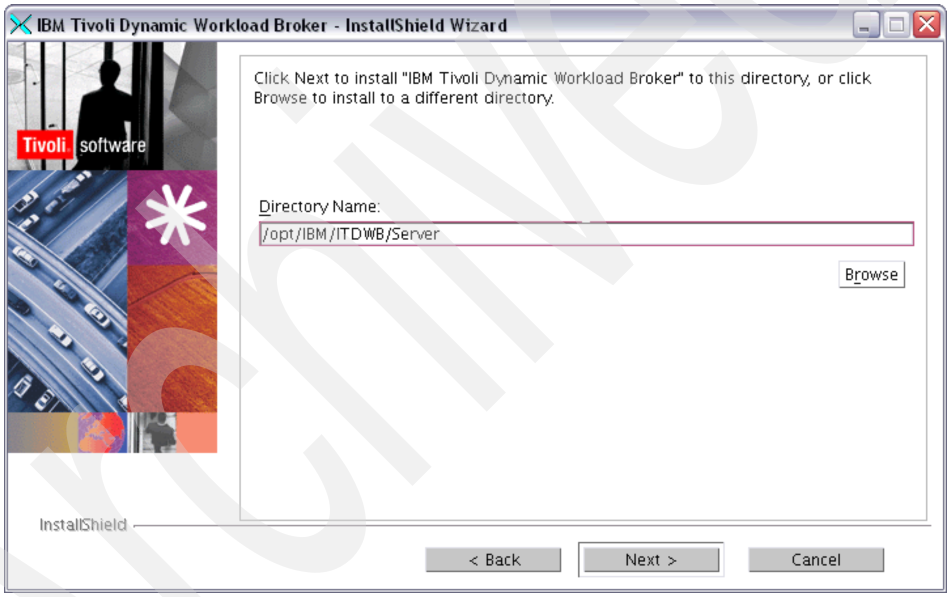
   – Install the IBM Tivoli Dynamic Workload Broker server
   – Install the IBM Tivoli Dynamic Workload Broker Web Console
   – Install the IBM Tivoli Dynamic Workload Broker Job Brokering Definition Console
   – Install the IBM Tivoli Dynamic Workload Broker agent

3.3.2 Installing the Tivoli Dynamic Workload Broker server with the installation wizard

Use the following steps to install any of the Tivoli Dynamic Workload Broker server components using the installation wizard:

1. Start the installation launchpad. Select the language in which you want the wizard to display, then click **OK**. The wizard displays the Welcome window.
2. Click **Next** to continue with the installation. You are then prompted to read and accept the license agreement. You also have the choice to print out the license agreement. To continue, check **I accept both the IBM and non-IBM terms**, then click **Next**.

   The installation directory window is displayed, as shown in Figure 3-3.



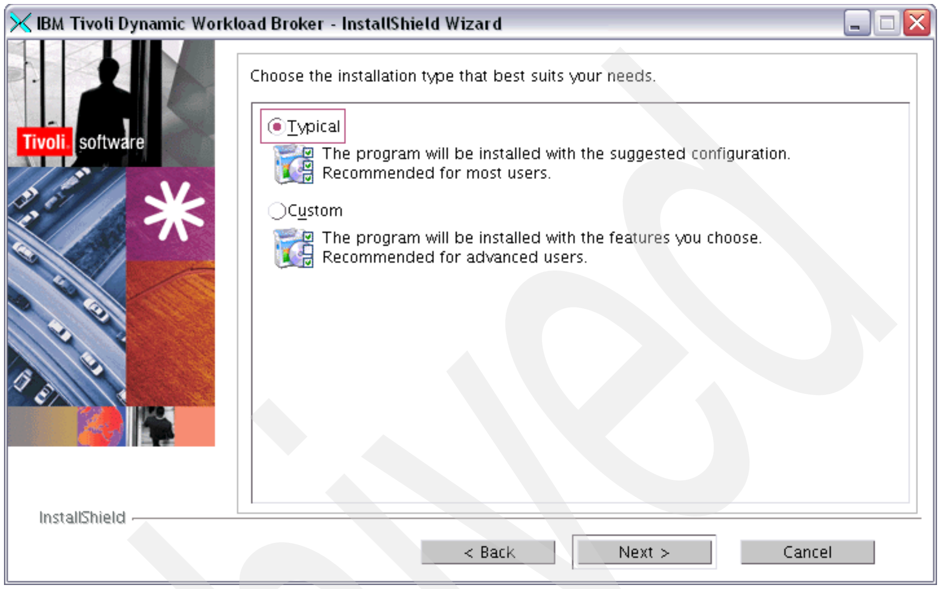
Click Next to install "IBM Tivoli Dynamic Workload Broker" to this directory, or click Browse to install to a different directory.

Directory Name: /opt/IBM/ITDWB/Server

Figure 3-3 The installation directory window

3. You can either accept the default directory displayed or click **Browse** to select your desired installation location. Once that is decided, click **Next** to continue.

   The installation choice window is displayed (Figure 3-4).



Choose the installation type that best suits your needs.

(●) Typical
The program will be installed with the suggested configuration. Recommended for most users.

( ) Custom
The program will be installed with the features you choose. Recommended for advanced users.

Figure 3-4 The installation choice window

Here is where you choose whether you would like a typical or custom install.

– Typical

  A typical installation installs every component required on the same computer, except for the following:

  - Enterprise Workload Manager enablement
  - Tivoli Workload Scheduler agent
  - IBM Change and Configuration Management Database enablement
  - Tivoli Provisioning Manager enablement

– Custom

  A custom installation allows you to select each component that you would like to install. You would use the custom installation if you want to install components on more than one computer.

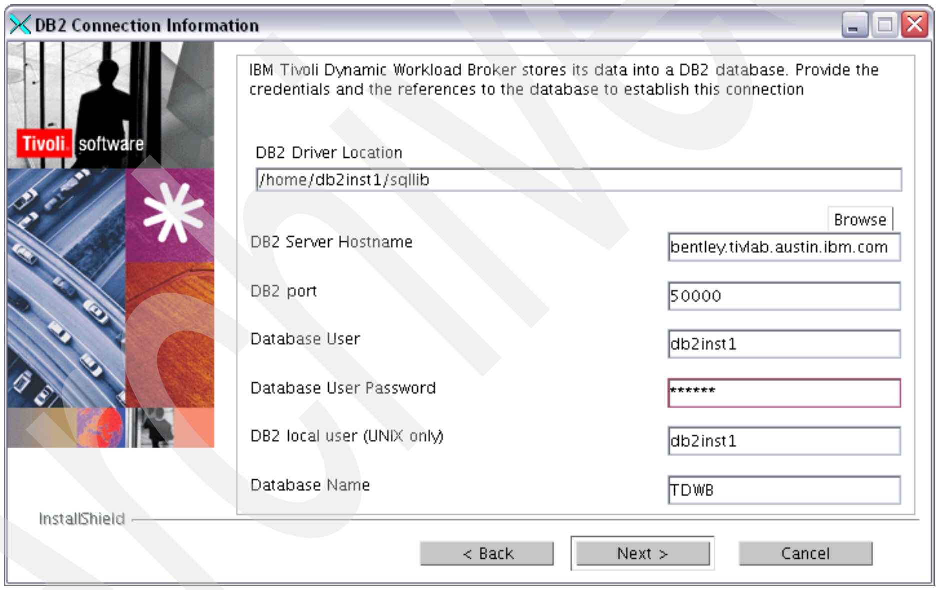
When using the typical installation option, the following features are always installed and configured:

– Tivoli Agent Manager
– Tivoli Agent Manager database
– IBM Tivoli Dynamic Workload Broker server
– IBM Tivoli Dynamic Workload Broker database

If you plan to install the Agent Manager before you install the Tivoli Dynamic Workload Broker server, or you want to install the Agent Manager on a separate computer, you would use the custom installation option.

We proceed with the Typical install option. Custom install is covered in "Custom install" on page 108.

4. Once the **Typical** option has been selected, the DB2 server information window is displayed, as shown in Figure 3-5.



IBM Tivoli Dynamic Workload Broker stores its data into a DB2 database. Provide the credentials and the references to the database to establish this connection

DB2 Driver Location	/home/db2inst1/sqllib
DB2 Server Hostname	bentley.tivlab.austin.ibm.com
DB2 port	50000
Database User	db2inst1
Database User Password	******
DB2 local user (UNIX only)	db2inst1
Database Name	TDWB

Figure 3-5 The DB2 server information window

You will need to fill out the following information:

– DB2 Driver Location

The directory where the client or server version of DB2 is installed.

- Windows: C:\Program Files\IBM\SQLLIB
- UNIX: /usr/...

- DB2 Server Hostname

  The name of the host that you are going to connect to DB2. The default is the fully qualified host name of the local computer. This can be either a DB2 server or client.

- DB2 Port

  The port on which DB2 will listen. The default is 50000.

- Database User and Database User Password

  The DB2 instance owner and password for an account to be created on this computer. The user ID that you enter here is used for connection to the DB2 database. The predefined user IDs are:

  - Windows: db2admin
  - UNIX: db2inst1

  Note: You can use a different ID. If you do, the user ID must exist in the DB2 database and must be configured in DB2 and on the DB2 server prior to this install. If this user is not an administrator, you will need to specify an administrator account on the next screen.

- DB2 local user (UNIX only)

  The user on the local operating system that is running the DB2 client binaries

- Database name

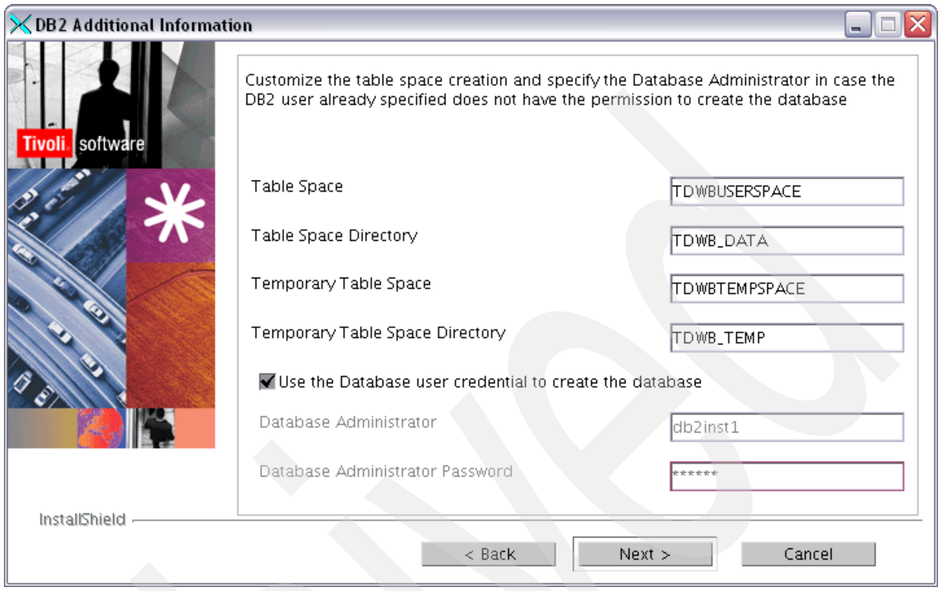
  The name of the Tivoli Dynamic Workload Broker database. There are restrictions to the length of the database name. For more information about this, refer to your DB2 manuals at:

  `http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp`

The Tivoli Dynamic Workload Broker database will be created on the computer where the DB2 server resides. The default database name of TDWB will be used unless otherwise specified. If the database exists, the existing database is used and the Database Name field is not displayed.

5. Once everything is filled out, click **Next**. A check is performed to establish a connection with the DB2 database server. If no connection can be made, an error message is displayed. Make sure that the DB2 database is started. When a connection to the DB2 database is established, the next window is displayed.

If a Tivoli Dynamic Workload Broker database has not previously been installed, the DB2 additional information window is displayed. See Figure 3-6.



Customize the table space creation and specify the Database Administrator in case the DB2 user already specified does not have the permission to create the database

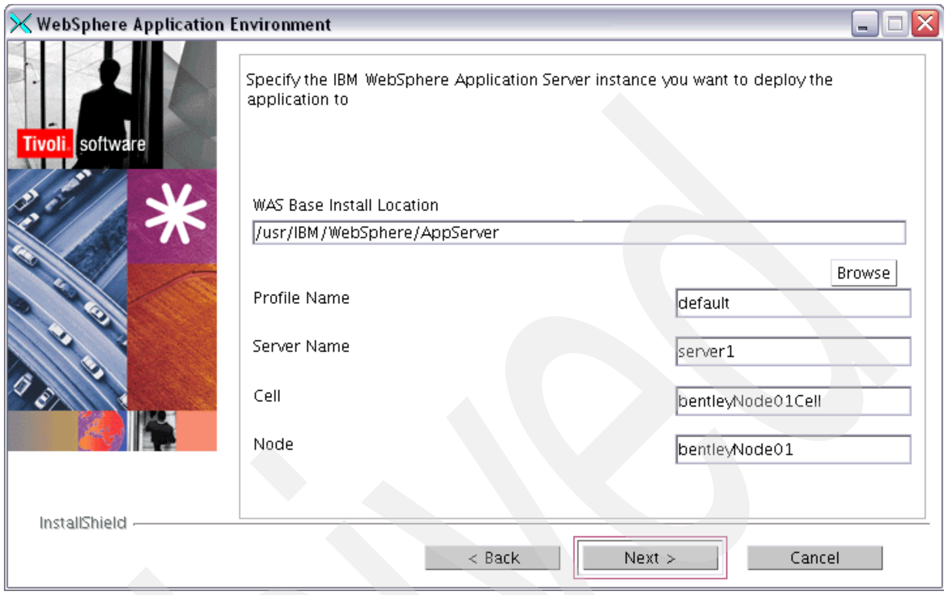
Table Space: TDWBUSERSPACE
Table Space Directory: TDWB_DATA
Temporary Table Space: TDWBTEMPSPACE
Temporary Table Space Directory: TDWB_TEMP
[✓] Use the Database user credential to create the database
Database Administrator: db2inst1
Database Administrator Password: ******

Figure 3-6 The DB2 additional information window

6. If required, specify the Table Space, Table Space Directory, Temporary Table Space, and Temporary Table Space Directory. Otherwise leave the default values.
7. If you want to change the DB2 user or have not specified an administrator in the previous panel, select the **Use the DB2 user credential to create the database** check box and type in the user name and password for the user to create the database. If the user you specified previously is an administrator, you can use this user. The user ID that you enter here is used for the connection to the DB2 database. The predefined user IDs are:
   - Windows: db2admin
   - UNIX: db2inst1

   You can use a different ID. If you do, the user ID must exist in the DB2 database and must be configured in DB2 and on the DB2 server prior to this install. Click **Next**.

The WebSphere Application environment window is displayed. See Figure 3-7.



Specify the IBM WebSphere Application Server instance you want to deploy the application to

WAS Base Install Location: /usr/IBM/WebSphere/AppServer

Profile Name: default

Server Name: server1

Cell: bentleyNode01Cell

Node: bentleyNode01

Figure 3-7 The WebSphere Application environment window

8. Check the following WebSphere information displayed in the window:
   - Base Install Location

     The directory where the WebSphere Application Server is installed.
   - Profile Name

     The existing name of a WebSphere Application Server profile. The profile used here must exist.
   - Server Name

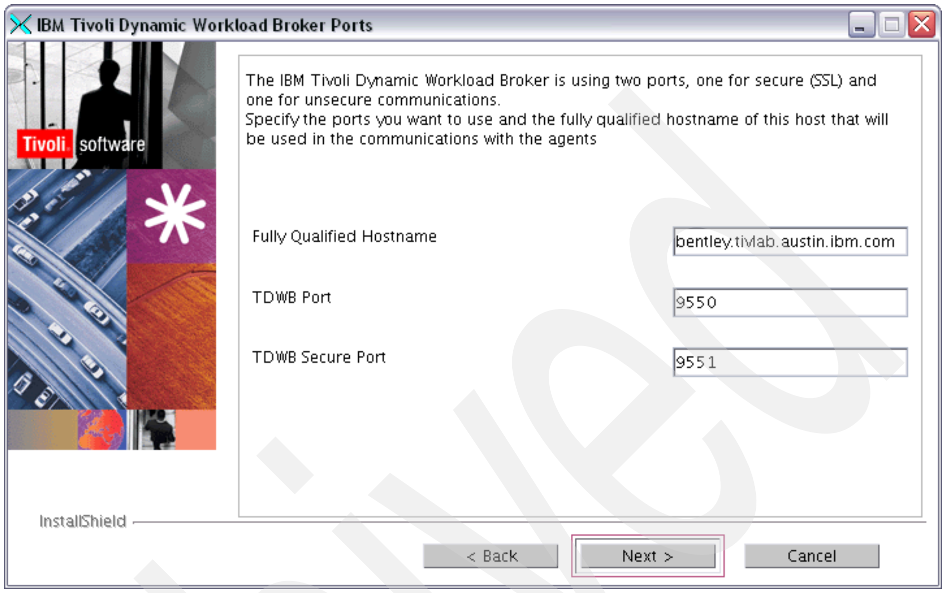
     The existing name of the WebSphere Application Server.
   - Cell

     The cell name. This is automatically discovered if a valid directory is specified for the Base Install Location.
   - Node

     The WebSphere node name. This is automatically discovered if a valid directory is specified for the base install location.

   Click **Next**.

The Tivoli Dynamic Workload Broker Ports window is displayed, as shown in Figure 3-8.



The IBM Tivoli Dynamic Workload Broker is using two ports, one for secure (SSL) and one for unsecure communications.
Specify the ports you want to use and the fully qualified hostname of this host that will be used in the communications with the agents

Fully Qualified Hostname: bentley.tivlab.austin.ibm.com

TDWB Port: 9550

TDWB Secure Port: 9551

Figure 3-8 The Tivoli Dynamic Workload Broker Ports window

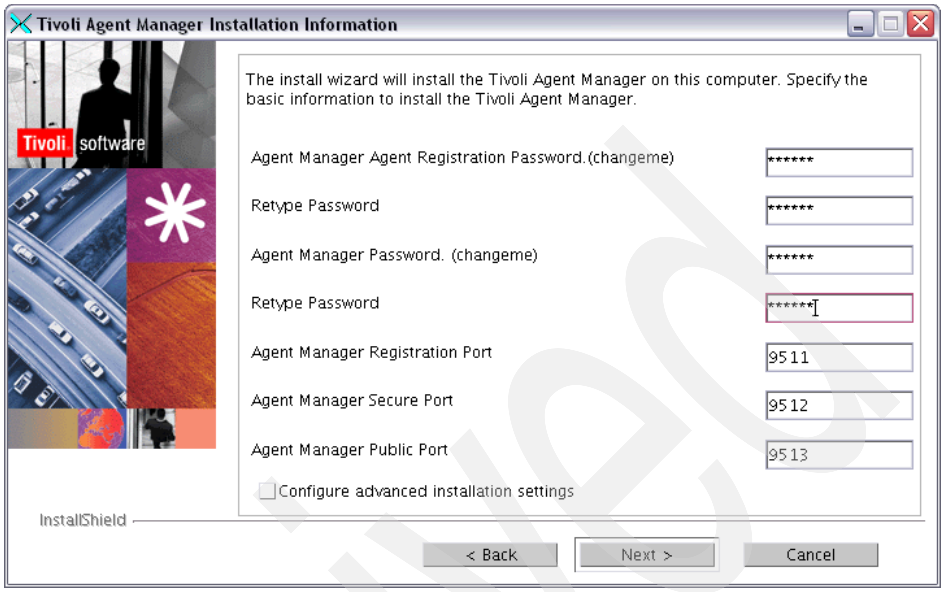
9. This contains the values for the two ports used by Tivoli Dynamic Workload Broker.
   - TDWB Port

     This value is used for insecure communications. The default is 9550.
   - TDWB Secure Port

     This value is used for secure (SSL) communications. The default is 9551.

   The default values can be changed if needed. Click **Next**.

The Agent Manager information window is displayed. See Figure 3-9.



The install wizard will install the Tivoli Agent Manager on this computer. Specify the basic information to install the Tivoli Agent Manager.

Agent Manager Agent Registration Password.(changeme): ******
Retype Password: ******
Agent Manager Password. (changeme): ******
Retype Password: ******
Agent Manager Registration Port: 9511
Agent Manager Secure Port: 9512
Agent Manager Public Port: 9513
Configure advanced installation settings

Figure 3-9 The Agent Manager information window

10. You must supply all of the following information in this window:

   – Agent Manager Agent Registration Password

     The default password is *changeme*. This password is presented by a common agent when it requests registration with the Agent Manager. This password also locks the agentTrust.jks truststore file, which contains the signer certificate for the Agent Manager.

   – Retype Password

     Retype the above password to confirm.

   – Agent Manager Password

     The default password is changeme. This is the Agent Manager's SSL password. This password locks the Agent Manager truststore and keystore files, agentManagerTrust.jks and agentManagerKeys.jks.

     Note: Agent Manager passwords cannot contain the following characters: > < " = ; , ^ / & | £ and the space character.

   – Retype Password

     Retype the above password to confirm.

– Agent Manager Registration Port

  The port number for registration. The default is 9511. This port uses server-side authentication.

– Agent Manager Secure Port

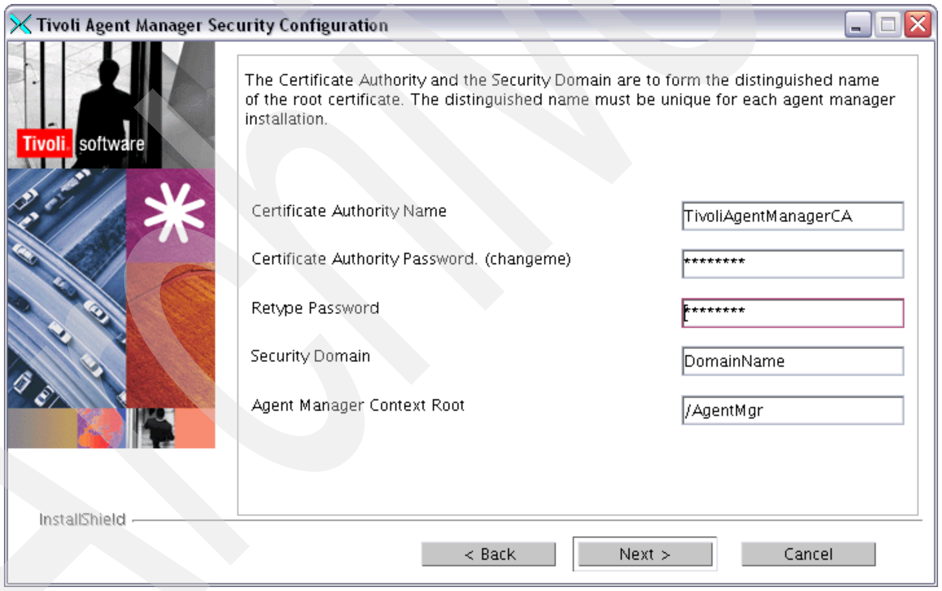
  The port number for secure communications with client authentication by 2-way SSL. The default is 9512.

– Agent Manager Public Port

  The port number for public communication, including the alternate port for the agent recovery service. The default is 9513.

If you need to configure advanced installation settings, check **Configure advanced installation settings**, then click **Next**.

11. If you checked **Configure advanced installation settings**, the Agent Manager advanced installation settings window is displayed, as shown in Figure 3-10.



The Certificate Authority and the Security Domain are to form the distinguished name of the root certificate. The distinguished name must be unique for each agent manager installation.

Certificate Authority Name	TivoliAgentManagerCA
Certificate Authority Password. (changeme)	********
Retype Password	********
Security Domain	DomainName
Agent Manager Context Root	/AgentMgr

Figure 3-10 The Agent Manager advanced installation settings window

You will need to supply the following information about this window:

– Certificate Authority Name

  The name of the certificate authority. This value must be unique in your environment.

– Certificate Authority Password

The password for the certificate authority.

– Security Domain

  The name of the security domain defined by the Agent Manager. The security domain is used in the right-hand portion of the distinguished name (DN) of every certificate issued by the Agent Manager. Typically, this value is the registered domain name or contains the registered domain name, but can be any value you chose. For example, for the computer system myserver.ibm.com, the domain name might be ibm.com. This value must be unique in your environment.

– Agent Manager Context Root

  The name of the certificate authority. This value must be unique in your environment. The default value is /AgentMgr. The context root is part of the URL that common agents and resource managers use to connect to the Agent Manager. For example, the context root is the underlined part of the following:

  `http://bentley.tivlab.austin.ibm.com:9513/AgentMgr`

Click **Next**.

12. If you checked **Configure advanced installation settings** the Agent Manager deployment information window is displayed. You will need to supply the following information on this window:

– Agent Manager host name

  The fully qualified host name of the Agent Manager

> **Note:** The Agent Manager host name must be resolved on the endpoint, even if you want to use the IP address to indicate the Agent Manager server. Ensure that the host file on the endpoint or the DNS server is properly configured to resolve the Agent Manager host.

– Cell

  Specifies the name of the WebSphere cell where the Agent Manager applications are installed

– Node

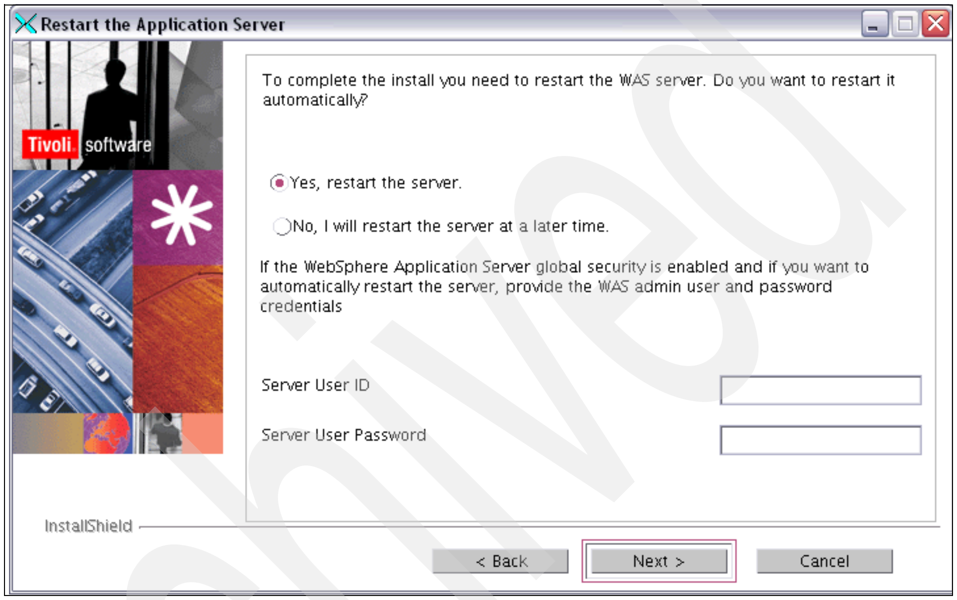
  Specifies the name of the WebSphere node where the Agent Manager applications are installed

– Server Name

  The name of the WebSphere Application Server

Click **Next**.

13. To complete the installation it is necessary to restart the WebSphere Application Server. A window is displayed (Figure 3-11) asking whether you want the server restarted automatically, or whether you want to restart the server yourself at a later time. If you want the server restarted, and if the WebSphere Application Server global security is enabled, you must provide the WebSphere Application Server administrator user ID and password. Select when you want the server restarted, then click **Next**.



To complete the install you need to restart the WAS server. Do you want to restart it automatically?

[✓ Yes, restart the server.]

No, I will restart the server at a later time.

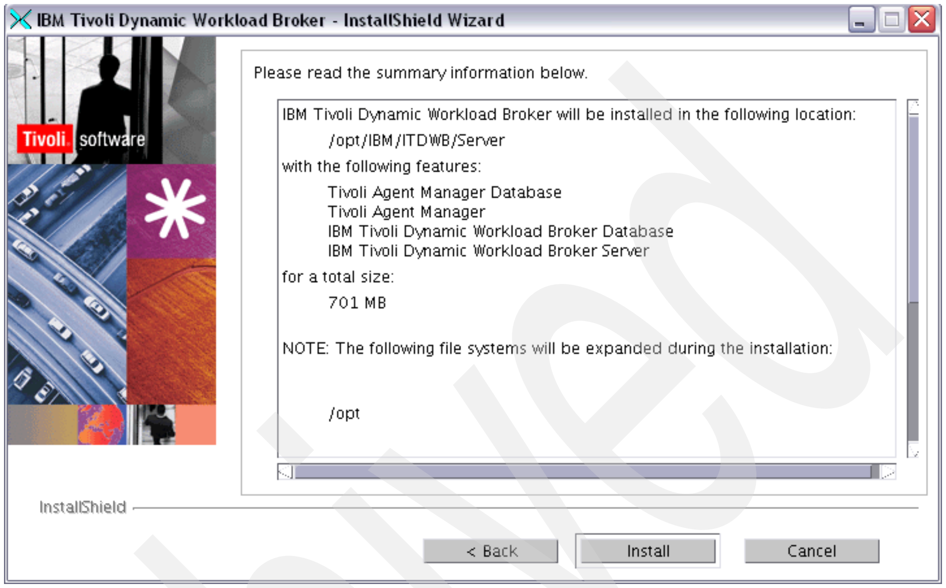
If the WebSphere Application Server global security is enabled and if you want to automatically restart the server, provide the WAS admin user and password credentials

Server User ID

Server User Password

Figure 3-11 Restart the Application Server

The Installation Summary window is displayed, as shown in Figure 3-12. Check the information displayed in the window. Ensure that there is enough disk space available for the installation.



Please read the summary information below.

```
IBM Tivoli Dynamic Workload Broker will be installed in the following location:
      /opt/IBM/ITDWB/Server
with the following features:
      Tivoli Agent Manager Database
      Tivoli Agent Manager
      IBM Tivoli Dynamic Workload Broker Database
      IBM Tivoli Dynamic Workload Broker Server
for a total size:
      701 MB

NOTE: The following file systems will be expanded during the installation:


      /opt
```

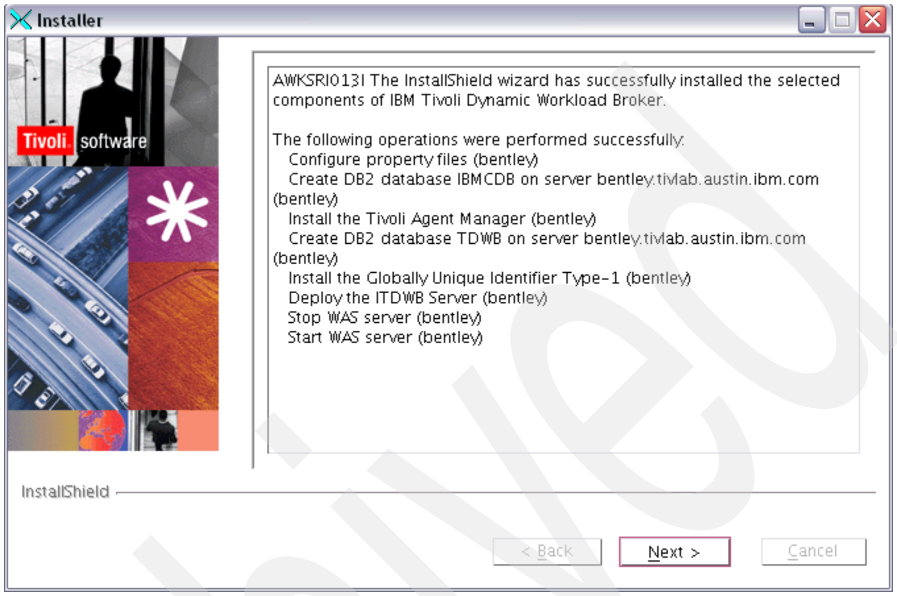
Figure 3-12 The Installation Summary window

14. Click **Install**.

15. When the installation progress window is displayed, if the uninstaller has not yet been created, you can stop the installation at any time by clicking **Cancel**. The installation will complete its current installation step, then suspend the installation. You will be asked whether you want to cancel the installation. If you choose **Yes**, the current installation is cancelled.

    After the uninstaller has been created, if you click **Stop** when the installation steps are in progress, you will be asked whether you want to stop the installation after the current step. If you click **Yes**, the step completes and the Diagnose Failure window is displayed. If you click **No**, the installation resumes.

When the installation has successfully completed, an installation summary screen (Figure 3-13) is displayed showing the operations that have been performed during the installation.

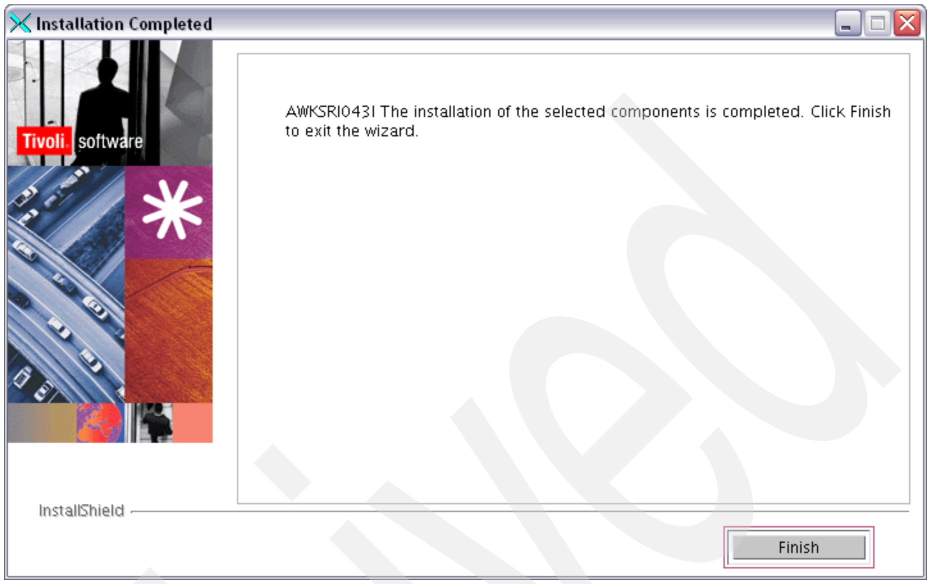


```
AWKSRI013I The InstallShield wizard has successfully installed the selected
components of IBM Tivoli Dynamic Workload Broker.

The following operations were performed successfully:
   Configure property files (bentley)
   Create DB2 database IBMCDB on server bentley.tivlab.austin.ibm.com
(bentley)
   Install the Tivoli Agent Manager (bentley)
   Create DB2 database TDWB on server bentley.tivlab.austin.ibm.com
(bentley)
   Install the Globally Unique Identifier Type-1 (bentley)
   Deploy the ITDWB Server (bentley)
   Stop WAS server (bentley)
   Start WAS server (bentley)
```

Figure 3-13 The installation has successfully completed

16. Click **Next**.

17. The installation completed window is displayed. Click **Finish** to close the installer.



AWKSRI043I The installation of the selected components is completed. Click Finish to exit the wizard.

Figure 3-14 The installation completed window

Custom install

When using the custom install option, you have the option to create the following features:

- Tivoli Agent Manager database
- Tivoli Agent Manager
- IBM Tivoli Dynamic Workload Broker database
- IBM Tivoli Dynamic Workload Broker server
- IBM Tivoli Dynamic Workload Broker extensions:
  - Enterprise Workload Manager enablement
  - Tivoli Workload Scheduler Agent
  - IBM Change and Configuration Management Database enablement
  - IBM Tivoli Provisioning Manager enablement

Note: The Agent Manager is installed in two steps:

1. Tivoli Agent Manager Database

   This step creates the database and tables.

2. Tivoli Agent Manager

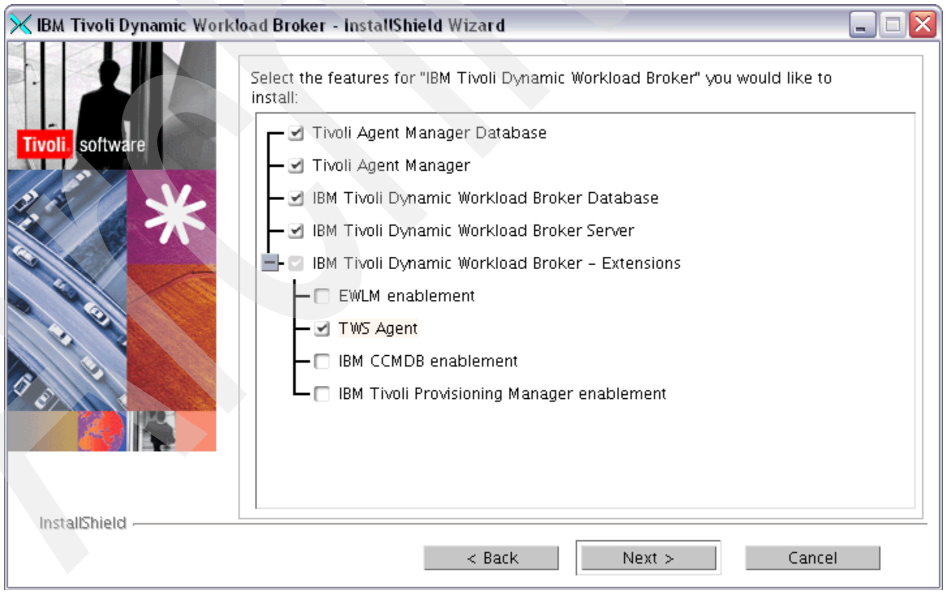
   This installs the enterprise application on WebSphere and creates references to the database.

   These steps must be performed in the above order for the Agent Manager installation. The installation fails if the Tivoli Agent Manager is installed before the Tivoli Agent Manager database. Also, the Tivoli Agent Manager is a prerequisite of Tivoli Dynamic Workload Broker and must be installed before or with the product.

The custom installation is similar to the typical installation, except that you have the choice of what Tivoli Dynamic Workload Broker features you want installed.

Complete the custom installation as follows:

1. When you select the custom installation, the features window is displayed. See Figure 3-15.



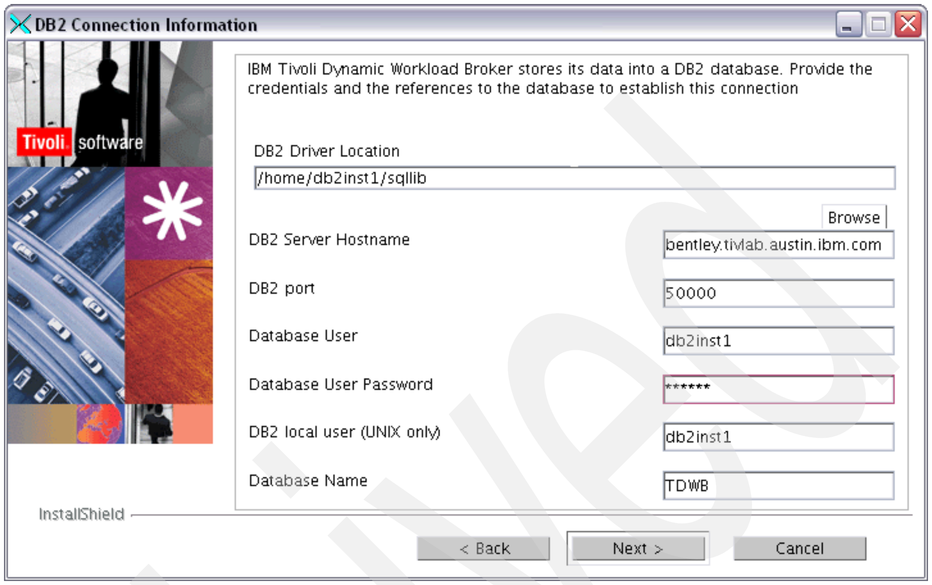
Select the features for "IBM Tivoli Dynamic Workload Broker" you would like to install:

- ✓ Tivoli Agent Manager Database
- ✓ Tivoli Agent Manager
- ✓ IBM Tivoli Dynamic Workload Broker Database
- ✓ IBM Tivoli Dynamic Workload Broker Server
- ✓ IBM Tivoli Dynamic Workload Broker – Extensions
  - ☐ EWLM enablement
  - ✓ TWS Agent
  - ☐ IBM CCMDB enablement
  - ☐ IBM Tivoli Provisioning Manager enablement

Figure 3-15   Custom installation

Select the components and extensions you want to install.

2. If you selected to install the Tivoli Dynamic Workload Broker Database, the DB2 server information window is displayed. See Figure 3-16.



IBM Tivoli Dynamic Workload Broker stores its data into a DB2 database. Provide the credentials and the references to the database to establish this connection

DB2 Driver Location: /home/db2inst1/sqllib

DB2 Server Hostname: bentley.tivlab.austin.ibm.com

DB2 port: 50000

Database User: db2inst1

Database User Password: ******

DB2 local user (UNIX only): db2inst1

Database Name: TDWB

Figure 3-16 The DB2 server information window

You will need to fill out the following information:

– DB2 Driver Location

  The directory where the client or server version of DB2 is installed.

  - Windows: C:\Program Files\IBM\SQLLIB
  - UNIX: /usr/...

– DB2 Server Hostname

  The name of the host that you are going to connect to DB2. The default is the fully qualified host name of the local computer. This can be either a DB2 server or client.

– DB2 Port

  The port on which DB2 will listen. The default is 50000.

– Database User and Database User Password

  The DB2 instance owner and password for an account to be created on this computer. The user ID that you enter here is used for connection to the DB2 database. The predefined user IDs are:

  - Windows: db2admin
  - UNIX: db2inst1

Note: You can use a different ID. If you do, the user ID must exist in the DB2 database and must be configured in DB2 and on the DB2 server prior to this install. If this user is not an administrator, you will need to specify an administrator account on the next screen.

- DB2 local user (UNIX only)

  The user on the local operating system that is running the DB2 client binaries.

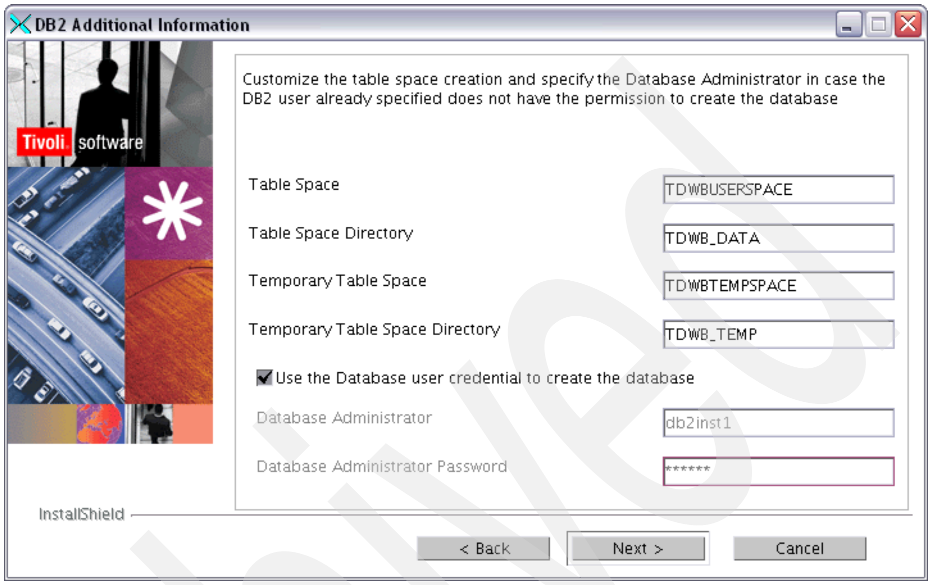
- Database name

  The name of the Tivoli Dynamic Workload Broker database. There are restrictions to the length of the database name. For more information about this, refer to your DB2 manuals.

  The Tivoli Dynamic Workload Broker database is created on the computer where the DB2 server resides. The default database name of TDWB is used unless otherwise specified. If the database exists, the existing database is used and the Database Name field is not displayed.

3. Once everything is filled out, click **Next**. A check is performed to establish a connection with the DB2 database server. If no connection can be made, an error message is displayed. Make sure that the DB2 database is started. When a connection to the DB2 database is established, the next window is displayed.

4. If a Tivoli Dynamic Workload Broker database has not previously been installed, the DB2 additional information window is displayed. See Figure 3-17.



Customize the table space creation and specify the Database Administrator in case the DB2 user already specified does not have the permission to create the database

Table Space	TDWBUSERSPACE
Table Space Directory	TDWB_DATA
Temporary Table Space	TDWBTEMPSPACE
Temporary Table Space Directory	TDWB_TEMP

[✓] Use the Database user credential to create the database

Database Administrator	db2inst1
Database Administrator Password	******

Figure 3-17 The DB2 additional information window

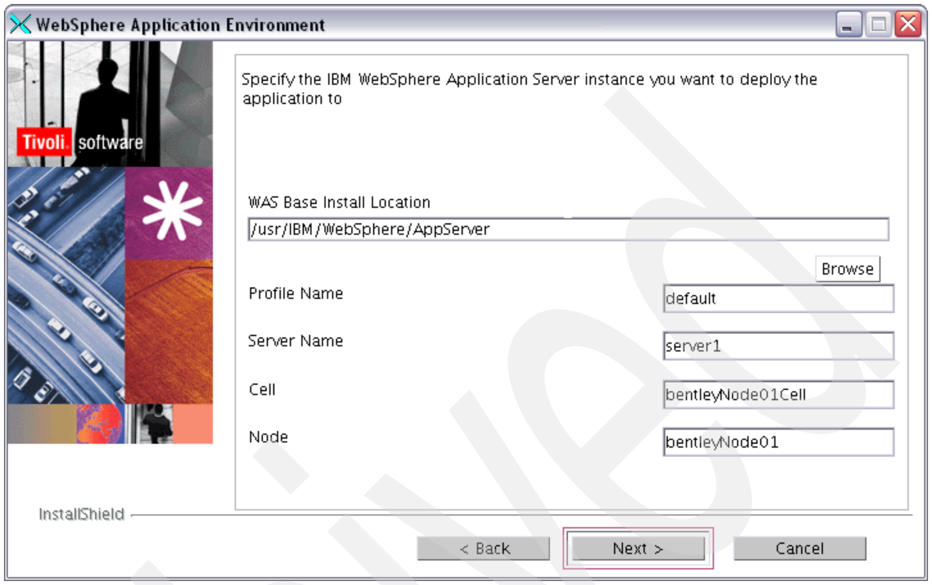
5. If required, specify the Table Space, Table Space Directory, Temporary Table Space, and Temporary Table Space Directory. Otherwise leave the default values.

   If you want to change the DB2 user or have not specified an administrator in the previous panel, select the **Use the DB2 user credential to create the database** check box and type in the user name and password for the user to create the database. If the user you specified previously is an administrator, you can use this user. The user ID that you enter here is used for the connection to the DB2 database. The predefined user IDs are:

   – Windows: db2admin
   – UNIX: db2inst1

   You can use a different ID. If you do, the user ID must exist in the DB2 database and must be configured in DB2 and on the DB2 server prior to this install. Click **Next**.

The WebSphere Application environment window is displayed. See Figure 3-18.



Specify the IBM WebSphere Application Server instance you want to deploy the application to

WAS Base Install Location: /usr/IBM/WebSphere/AppServer

Profile Name: default

Server Name: server1

Cell: bentleyNode01Cell

Node: bentleyNode01

Figure 3-18 The WebSphere Application environment window

6. Check the following WebSphere information displayed in the window:
   - Base Install Location

     The directory where the WebSphere Application Server is installed.
   - Profile Name

     The existing name of a WebSphere Application Server profile. The profile used here must exist.
   - Server Name

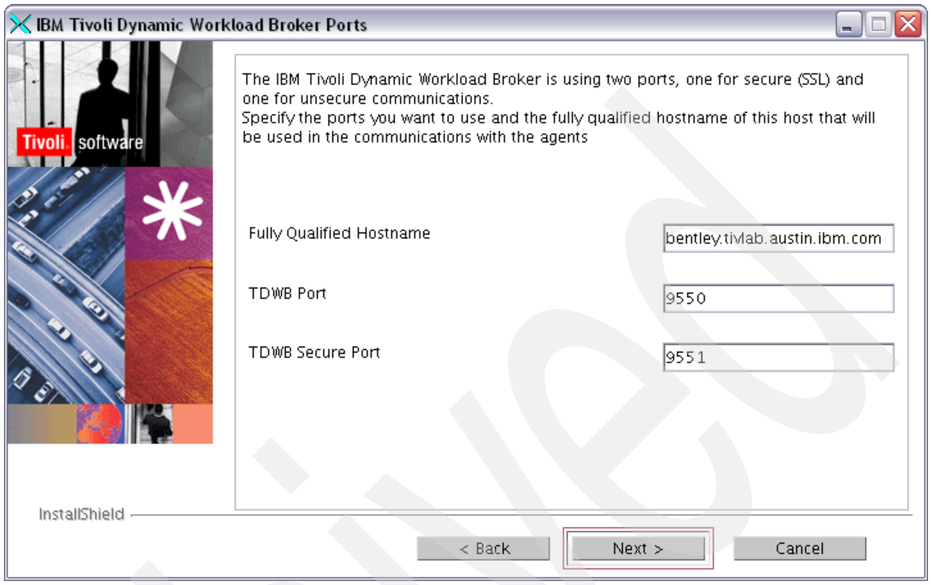
     The existing name of the WebSphere Application Server.
   - Cell

     The cell name. This is automatically discovered if a valid directory is specified for the base install location.
   - Node

     The WebSphere node name. This is automatically discovered if a valid directory is specified for the base install location.

   Click **Next**.

The Tivoli Dynamic Workload Broker Ports window is displayed, as shown in Figure 3-19.



The IBM Tivoli Dynamic Workload Broker is using two ports, one for secure (SSL) and one for unsecure communications.
Specify the ports you want to use and the fully qualified hostname of this host that will be used in the communications with the agents

Fully Qualified Hostname: bentley.tivlab.austin.ibm.com

TDWB Port: 9550

TDWB Secure Port: 9551

Figure 3-19 The Tivoli Dynamic Workload Broker Ports window

This window contains the values for the two ports used by Tivoli Dynamic Workload Broker.

– TDWB Port

  This value is used for unsecure communications. The default is 9550.

– TDWB Secure Port

  This value is used for secure (SSL) communications. The default is 9551.

The default values can be changed if needed.

Click **Next**.

7. If you selected to install the Tivoli Workload Scheduler Agent, the Tivoli Scheduler Workload Agent configuration window is displayed.

   Verify the information displayed in the window:

   – TWS Workstation Name

     Type the Tivoli Workload Scheduler name of this workstation. The name cannot exceed 16 characters and cannot contain spaces. You can use the default name suggested.

- TWS Master Domain Manager Name

  The workstation name of the Tivoli Workload Scheduler master domain manager. The name cannot exceed 16 characters and cannot contain spaces. This is a required field.

- TWS Agent Port

  The TCP/IP port number used by the instance being installed. The value must be in the range of 1–65535. The default value is 31111.

8. If you selected to install the Tivoli Workload Scheduler Agent, the TWS Agent Security Configuration window is displayed.

   - ITDWB User Name

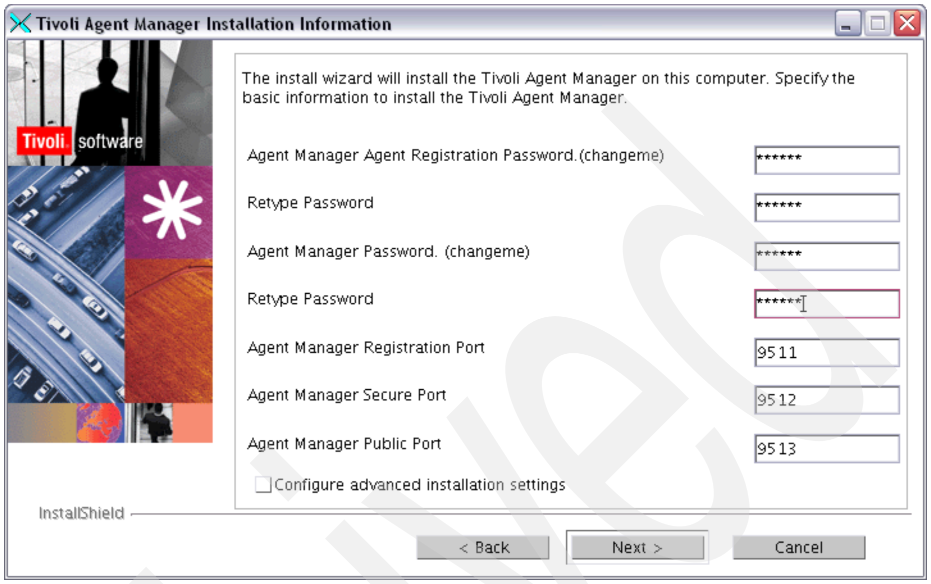
     The user that is used by the Tivoli Workload Scheduler agent when contacting the Tivoli Dynamic Workload Broker. The user must have at least the submitter role. This user must already be defined in WebSphere at installation time and also in the user registry. When you have completed the installation, you must give this user at least a submitter role.

     The user name entered here is written to the properties file. If required, you can change the user at a later time by editing the properties file.

   - ITDWB User Password

     The password for the ITDWB user.

9. If you selected to install the Tivoli Agent Manager, the Agent Manager information window is displayed.



The install wizard will install the Tivoli Agent Manager on this computer. Specify the basic information to install the Tivoli Agent Manager.

Agent Manager Agent Registration Password.(changeme)	******
Retype Password	******
Agent Manager Password. (changeme)	******
Retype Password	******
Agent Manager Registration Port	9511
Agent Manager Secure Port	9512
Agent Manager Public Port	9513

☐ Configure advanced installation settings

Figure 3-20 The Agent Manager information window

You must supply all of the following information in this window:

– Agent Manager Agent Registration Password

  The default password is changeme. This password is presented by a common agent when it requests registration with the Agent Manager. This password also locks the agentTrust.jks truststore file, which contains the signer certificate for the Agent Manager.

– Retype Password

  Retype the above password to confirm.

– Agent Manager Password

  The default password is changeme. This is the Agent Manager's SSL password. This password locks the Agent Manager truststore and keystore files, agentManagerTrust.jks and agentManagerKeys.jks.

  Note: Agent Manager passwords cannot contain the following characters: > < " = ; , ^ / & | £ and the space character.

– Retype Password

  Retype the above password to confirm.

– Agent Manager Registration Port

  The port number for registration. The default is 9511. This port uses server-side authentication.

– Agent Manager Secure Port

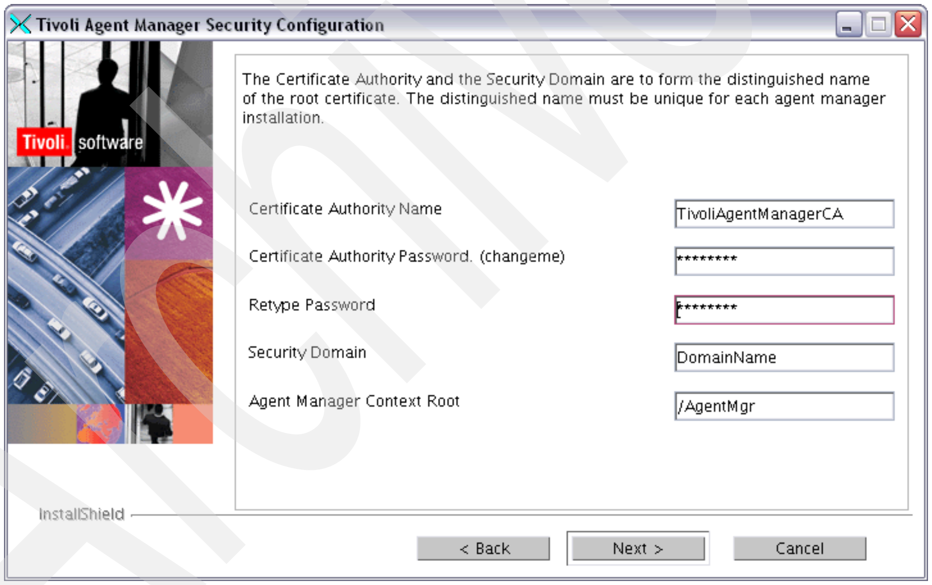
  The port number for secure communications with client authentication by 2-way SSL. The default is 9512.

– Agent Manager Public Port

  The port number for public communication, including the alternate port for the agent recovery service. The default is 9513.

If you need to configure advanced installation settings, check **Configure advanced installation settings**, then click **Next**.

10. If you selected to install the Tivoli Agent Manager, and you checked **Configure advanced installation settings**, the Agent Manager advanced installation settings window is displayed. See Figure 3-21.



The Certificate Authority and the Security Domain are to form the distinguished name of the root certificate. The distinguished name must be unique for each agent manager installation.

Certificate Authority Name	TivoliAgentManagerCA
Certificate Authority Password. (changeme)	********
Retype Password	********
Security Domain	DomainName
Agent Manager Context Root	/AgentMgr

Figure 3-21 the Agent Manager advanced installation settings window

You will need to supply the following information on this window:

– Certificate Authority Name

  The name of the certificate authority. This value must be unique in your environment.

– Certificate Authority Password

  The password for the certificate authority.

– Security Domain

  The name of the security domain defined by the Agent Manager. The security domain is used in the right-hand portion of the distinguished name (DN) of every certificate issued by the Agent Manager. Typically, this value is the registered domain name or contains the registered domain name, but can be any value you chose. For example, for the computer system myserver.ibm.com, the domain name might be ibm.com. This value must be unique in your environment.

– Agent Manager Context Root

  The name of the certificate authority. This value must be unique in your environment. The default value is /AgentMgr. The context root is part of the URL that common agents and resource managers use to connect to the Agent Manager. For example, the context root is the underlined part of the following:

  ```
  http://pluto.rome.ibm.com:9513/AgentMgr
  ```

Click **Next**.

11. If you selected to install the Tivoli Agent Manager, and you checked **Configure advanced installation settings**, the Agent Manager deployment information window is displayed. You will need to supply the following information on this window:

   – Agent Manager host name

     The fully qualified host name of the Agent Manager.

     Note: The Agent Manager host name must be resolved on the endpoint, even if you want to use the IP address to indicate the Agent Manager server. Ensure that the host file on the endpoint or the DNS server are properly configured to resolve the Agent Manager host.

   – Profile Directory

     Specifies the directory of an existing profile that defines the WebSphere Application Server run time.

   – Cell

     Specifies the name of the WebSphere cell where the Agent Manager applications are installed.

   – Node

Specifies the name of the WebSphere node where the Agent Manager applications are installed.

– Server Name

  The name of the WebSphere Application Server.

Click **Next**.

12. If you selected to install the IBM Enterprise Workload Manager (EWLM) feature, the EWLM enablement configuration window is displayed. You will need to supply the following information for the EWLM enablement configuration:

– EWLM Domain Manager Name

  The domain manager name of the Enterprise Workload Manager.

– EWLM Domain Manager Address

  The fully qualified address of the Enterprise Workload Manager domain manager.

– EWLM Domain Manager Port (LBP)

  The port for the Enterprise Workload Manager

– EWLM weight scope

  The weight scope for the Enterprise Workload Manager. This can be set to:

  - Application

    When calculating weight scopes, the Enterprise Workload Manager considers how the application is performing and takes into account CPU usage and system metrics. This is the default, as TDWB is instrumented to be monitored by EWLM.

  - System

    When calculating weight scopes, Enterprise Workload Manager considers CPU usage and system metrics only.

– EWLM refresh interval (sec)

  The refresh interval for the synchronization between the Tivoli Dynamic Workload Broker network and computers registered to Enterprise Workload Manager, in seconds. If a new computer is detected in the Tivoli Dynamic Workload Broker network, the computer is added to the Enterprise Workload Manager at the next synchronization. Similarly, if a computer is removed from the Tivoli Dynamic Workload Broker network, the computer is removed from the Enterprise Workload Manager at the next synchronization.

Click **Next**.

13. If you selected to install the IBM Change and Configuration Management Database (CCMDB) enablement feature, the CCMDB Configuration window is displayed. You need to supply the following information for the CCMDB enablement configuration:

    – CCMDB Server Hostname

      The fully qualified host name of the CCMDB server

    – CCMDB Server Port

      The port of the CCMDB server

    – CCMDB User

      The CCMDB user

    – CCMDB password

      The password for the CCMDB user

    Click **Next**.

14. If you selected to install the IBM Tivoli Provisioning Manager enablement feature, the Tivoli Provisioning Manager Configuration window is displayed. You need to supply the following information for the Tivoli Provisioning Manager enablement configuration:

    – TPM Server Hostname

      The fully qualified host name or the IP address of the TPM server

    – TPM Server port

      The port of the TPM server

    – TPM User

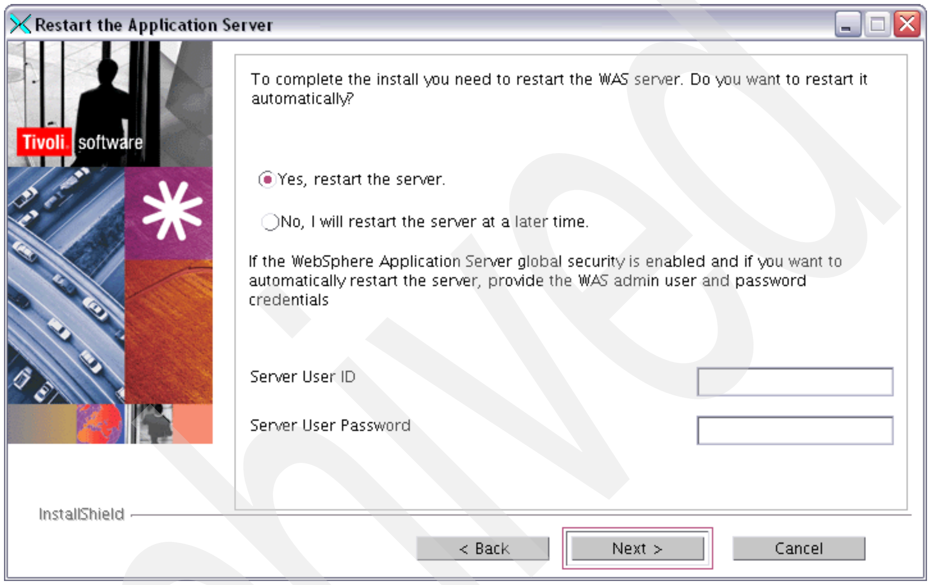
      The TPM user

    – TPM password

      The password for the TPM user

    Click **Next**.

15. To complete the installation it is necessary to restart the WebSphere Application Server. A window is displayed asking whether you want the server restarted automatically, or whether you want to restart the server yourself at a later time. If you want the server restarted, and if the WebSphere Application Server global security is enabled, you must provide the WebSphere Application Server administrator user ID and password. Select when you want the server restarted, then click **Next**. See Figure 3-22.



To complete the install you need to restart the WAS server. Do you want to restart it automatically?

[✓ Yes, restart the server.]

No, I will restart the server at a later time.

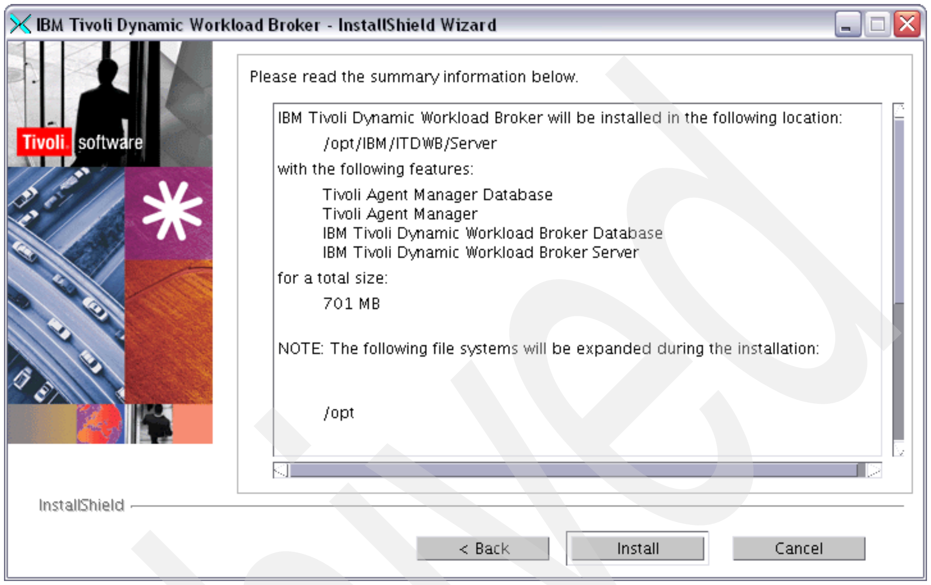
If the WebSphere Application Server global security is enabled and if you want to automatically restart the server, provide the WAS admin user and password credentials

Server User ID

Server User Password

Figure 3-22 Restart the Application Server

16. The Installation Summary window is displayed. Check the information displayed in the window. Ensure that there is enough disk space available for the installation.



Please read the summary information below.

IBM Tivoli Dynamic Workload Broker will be installed in the following location:

/opt/IBM/ITDWB/Server

with the following features:

Tivoli Agent Manager Database
Tivoli Agent Manager
IBM Tivoli Dynamic Workload Broker Database
IBM Tivoli Dynamic Workload Broker Server

for a total size:

701 MB

NOTE: The following file systems will be expanded during the installation:

/opt

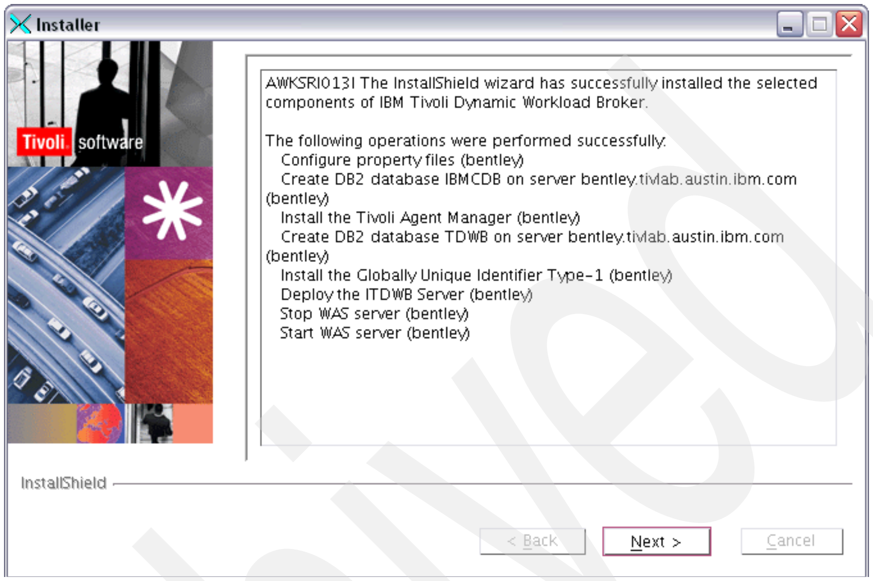
Figure 3-23 The Installation Summary window

Click **Install**.

17. When the installation progress window is displayed, if the uninstaller has not yet been created, you can stop the installation at any time by clicking **Cancel**. The installation will complete its current installation step, then suspend the installation. You will be asked if you want to cancel the installation. If you choose **Yes**, the current installation is cancelled.

After the uninstaller has been created, if you click **Stop** when the installation steps are in progress, you will be asked if you want to stop the installation after the current step. If you click **Yes**, the step completes and the Diagnose Failure window is displayed. If you click **No**, the installation resumes.

When the installation has successfully completed, an installation summary screen is displayed showing the operations that have been performed during the installation. See Figure 3-24.

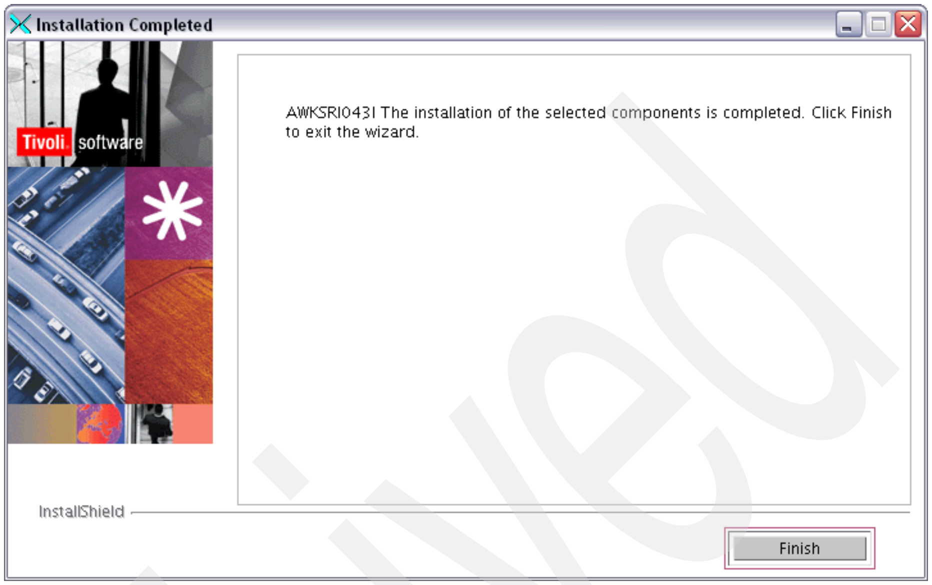


```
AWKSRI013I The InstallShield wizard has successfully installed the selected
components of IBM Tivoli Dynamic Workload Broker.

The following operations were performed successfully.
   Configure property files (bentley)
   Create DB2 database IBMCDB on server bentley.tivlab.austin.ibm.com
(bentley)
   Install the Tivoli Agent Manager (bentley)
   Create DB2 database TDWB on server bentley.tivlab.austin.ibm.com
(bentley)
   Install the Globally Unique Identifier Type-1 (bentley)
   Deploy the ITDWB Server (bentley)
   Stop WAS server (bentley)
   Start WAS server (bentley)
```

Figure 3-24 Successful installation

Click **Next**.

18. The Installation Completed window is displayed. Click **Finish** to close the installer (Figure 3-25).



AWKSRI043I The installation of the selected components is completed. Click Finish to exit the wizard.

Figure 3-25 Installation completion

3.3.3 Installing the Tivoli Dynamic Workload Broker Web Console

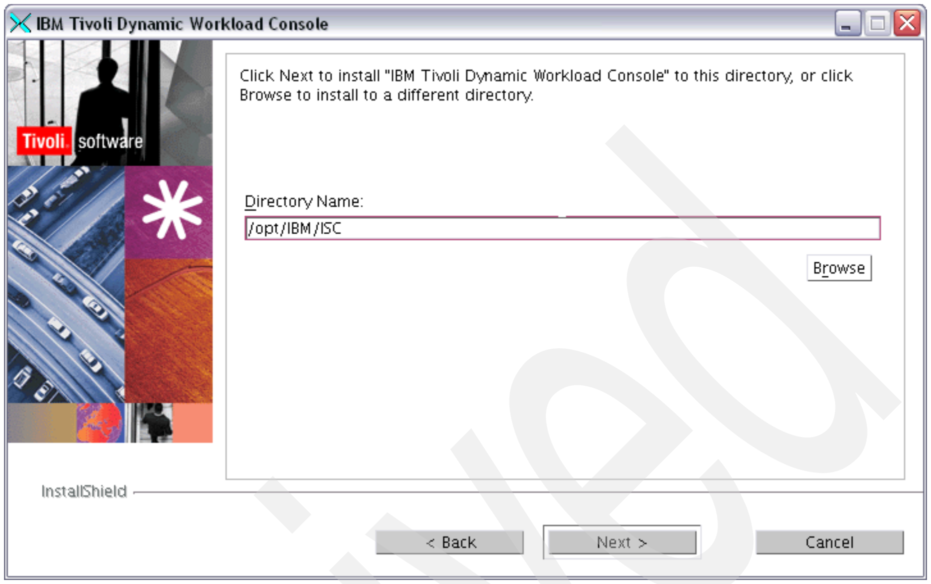
Perform the following to install the Tivoli Dynamic Workload Broker Web Console:

1. Via the launchpad, click **Install the IBM Tivoli Dynamic Workload Broker Web Console**. The introduction window is displayed. Click **Next** to continue with the installation. The license agreement is displayed. You will then be prompted to read and accept the license agreement. You also have the choice to print out the license agreement. To continue, check **I accept both the IBM and non-IBM terms**, then click **Next**.
2. The install then attempts to find whether an Integrated Solutions Console is already installed. If one is found, a window is displayed to tell you that it will be used to install the Tivoli Dynamic Workload Broker Web Console. If a console cannot be found, a window is displayed stating:

   ```
   No already installed stand-alone Integrated Solutions Console has
   been found. A new stand-alone Integrated Solutions Console is going
   to be installed for Tivoli Dynamic Workload Broker Web Console.
   ```

   Click **Next**.

3. The installation directory window is displayed, as shown in Figure 3-26.



Click Next to install "IBM Tivoli Dynamic Workload Console" to this directory, or click Browse to install to a different directory.

Directory Name:
/opt/IBM/ISC

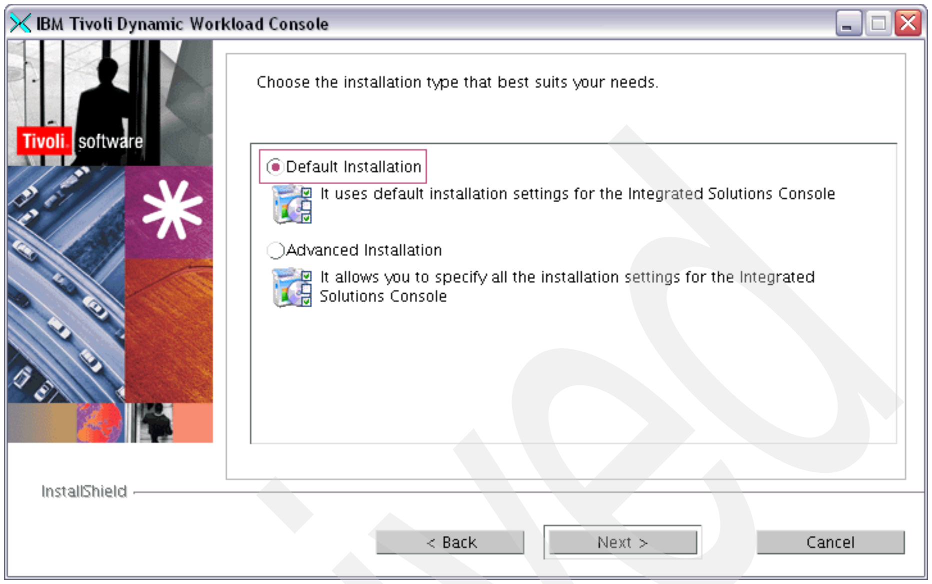
Figure 3-26 The installation directory window

4. You can either keep the default as listed, or click **Browse** to select a directory or drive of your choice. Click **Next** to accept the directory shown.

Note: Restrictions when installing the Integrated Solutions Console:

- If an existing directory is selected, the directory cannot contain any of the following files and directories:
  - The files product.reg and isc.properties
  - The directory /_uninst or a file name _uninst
  - The directory /AppServer or a file named AppServer
- The length of the installation path must be 32 characters or less.
- Only 127 ASCII characters are supported for the installation path.

5. The installation choice window is displayed, as shown in Figure 3-27.


IBM Tivoli Dynamic Workload Console

Choose the installation type that best suits your needs.

[✓ Default Installation]
It uses default installation settings for the Integrated Solutions Console

Advanced Installation
It allows you to specify all the installation settings for the Integrated Solutions Console

Figure 3-27 The installation choice window

Select the method you prefer from the options:

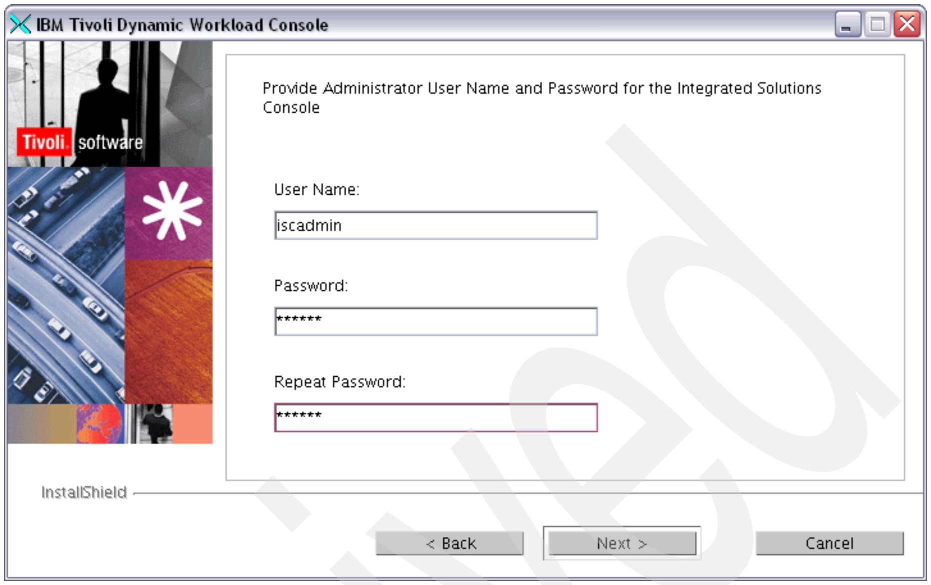
– Default installation
– Advanced installation

Tivoli Dynamic Workload Broker Web Console - default installation

With the default installation, there is no need to customize. The default installation is as follows:

1. Once the default installation is chosen, the Tivoli Dynamic Workload Broker Web Console host name window is displayed. Specify the host name and the ports to be used as the default connection, or leave the default values. Click **Next**.
2. To complete the installation, it is necessary to restart the ISC server. A window is displayed with the option of automatically restarting the server. Make your selection, then click **Next**.

The user name and password window is displayed (Figure 3-28).


IBM Tivoli Dynamic Workload Console

Provide Administrator User Name and Password for the Integrated Solutions Console

User Name: iscadmin

Password: ******

Repeat Password: ******

Figure 3-28 The user name and password window

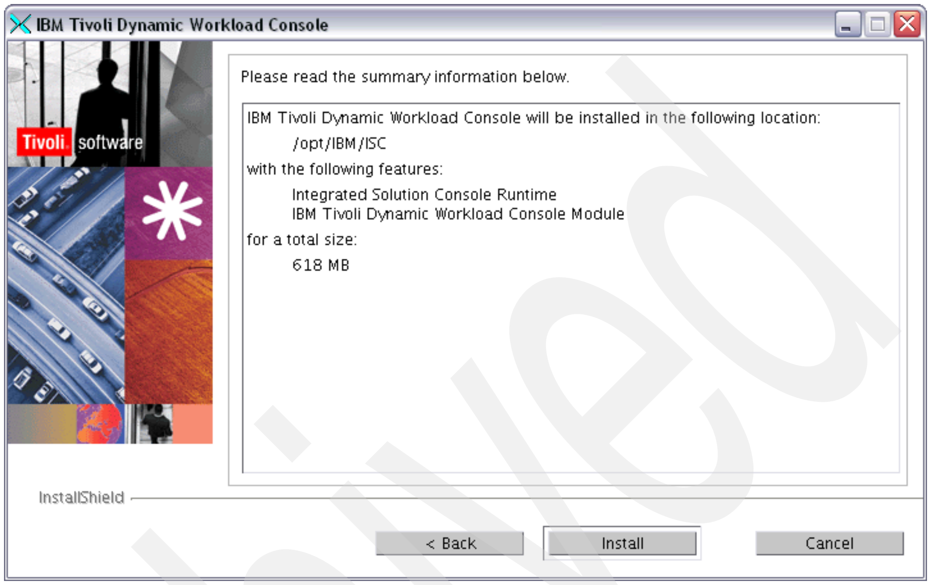
3. You can use an existing user, or have the installation create one for you. Either way, type the password, then type the password again for confirmation.

Note: The password can only contain the following characters:

- a–z
- A–Z
- 0–9
- . (period)
- - (hyphen)
- _ (underscore)

Click **Next**.

4. The verifying installation parameters window is displayed. Once the parameters have been verified, the installation summary window is displayed, as shown in Figure 3-29.



Please read the summary information below.

IBM Tivoli Dynamic Workload Console will be installed in the following location:

/opt/IBM/ISC

with the following features:

Integrated Solution Console Runtime
IBM Tivoli Dynamic Workload Console Module

for a total size:

618 MB

Figure 3-29 The verifying installation parameters window

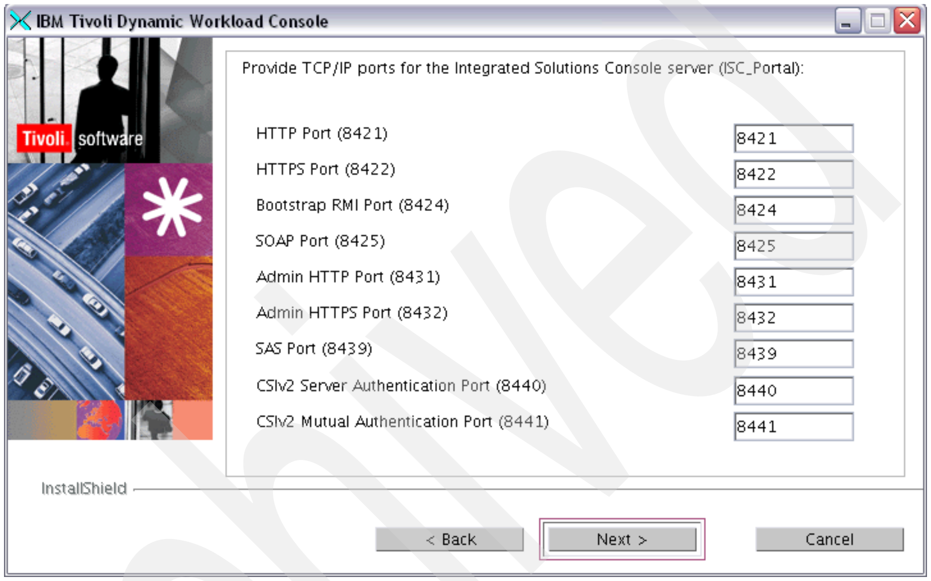
Click **Install**.

5. When the installation progress window is displayed, you can stop the installation at any time by clicking **Cancel**. The installation will complete its current installation step, then suspend the installation. You will be asked if you want to cancel the installation. If you choose **Yes**, the current installation is cancelled and the installation summary window is displayed showing the reason for the cancellation.
6. When the installation has successfully completed, an installation summary screen is displayed showing the operations that have been performed during the installation.
7. When the installation has completed successfully, and the installation summary window is displayed, take note of the URLs displayed on this panel. You will need this information to connect to the ISC later. Click **Next**. The installation completed window is displayed. Click **Finish** to close the installer.

Tivoli Dynamic Workload Broker Web Console - advanced installation

You can use the advanced installation when you need to customize any of the installation parameters.

1. When you select the advanced installation, the ISC TCP/IP ports window is displayed. See Figure 3-30.



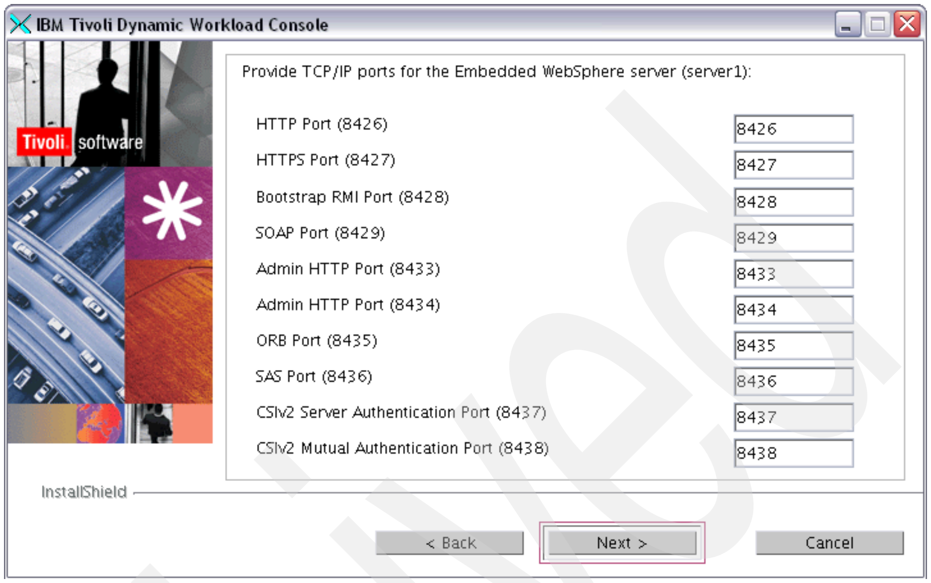
Provide TCP/IP ports for the Integrated Solutions Console server (ISC_Portal):

HTTP Port (8421)	8421
HTTPS Port (8422)	8422
Bootstrap RMI Port (8424)	8424
SOAP Port (8425)	8425
Admin HTTP Port (8431)	8431
Admin HTTPS Port (8432)	8432
SAS Port (8439)	8439
CSIv2 Server Authentication Port (8440)	8440
CSIv2 Mutual Authentication Port (8441)	8441

Figure 3-30 The ISC TCP/IP ports window

Specify the ports that you want to use for the operations console or accept the default values. The default value for each port is shown in parentheses beside the port name. Click **Next**.

The embedded WebSphere Application Server TCP/IP ports window is displayed, as shown in Figure 3-31.



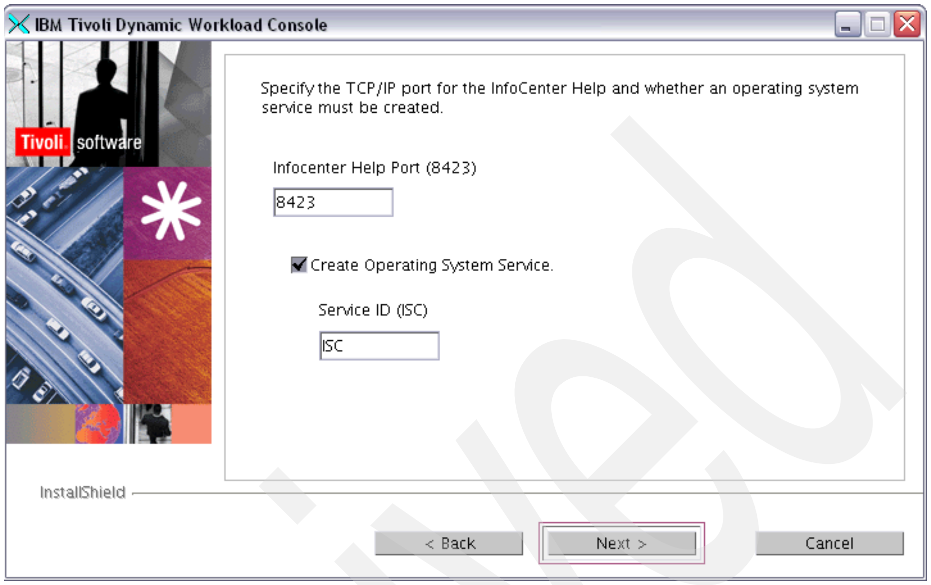
Provide TCP/IP ports for the Embedded WebSphere server (server1):

HTTP Port (8426)	8426
HTTPS Port (8427)	8427
Bootstrap RMI Port (8428)	8428
SOAP Port (8429)	8429
Admin HTTP Port (8433)	8433
Admin HTTP Port (8434)	8434
ORB Port (8435)	8435
SAS Port (8436)	8436
CSIv2 Server Authentication Port (8437)	8437
CSIv2 Mutual Authentication Port (8438)	8438

Figure 3-31 The embedded WebSphere Application Server TCP/IP ports window

2. Specify the ports that you want to use for the operations console or accept the default values. The default value for each port is shown in parentheses beside the port name. Click **Next**.

The InfoCenter TCP/IP port and service window is displayed (Figure 3-32).



Specify the TCP/IP port for the InfoCenter Help and whether an operating system service must be created.

Infocenter Help Port (8423)

8423

☑ Create Operating System Service.

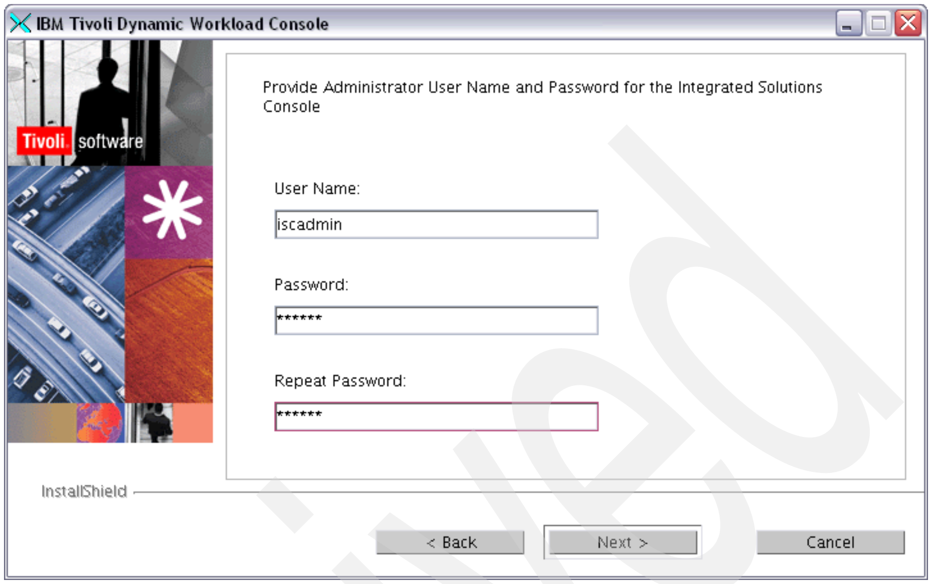
Service ID (ISC)

ISC

Figure 3-32 The InfoCenter TCP/IP port and service window

3. Specify the port for the InfoCenter help, then check if you want to register the ISC and the InfoCenter Help as operating system services. If you do, type the name for the ISC service and for the InfoCenter Help service, or accept the default values. Click **Next**.
4. The Tivoli Dynamic Workload Broker Web Console default connection window is displayed. Specify the host name and the port to be used as the default connection, or leave the default values. Click **Next**.
5. To complete the installation, it is necessary to restart the ISC server. A window is displayed with the option of automatically restarting the server. Make your selection, then click **Next**.

The user name and password window is displayed (Figure 3-33).


IBM Tivoli Dynamic Workload Console

Provide Administrator User Name and Password for the Integrated Solutions Console

User Name: iscadmin

Password: ******

Repeat Password: ******

Figure 3-33 The user name and password window

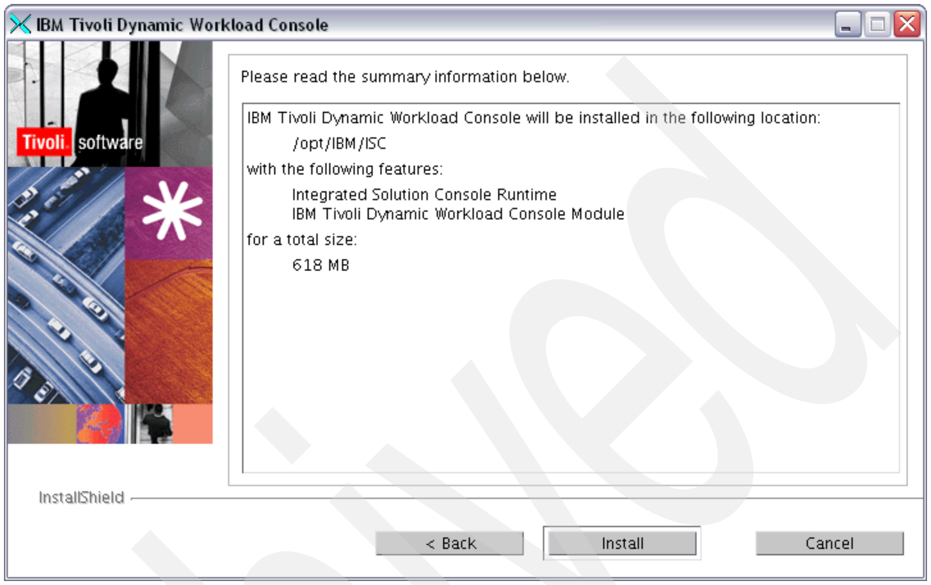
6. You can use an existing user, or have the installation create one for you. Either way, type the password, then type the password again for confirmation.

Note: The password can only contain the following characters:

- a–z
- A–Z
- 0–9
- . (period)
- - (hyphen)
- _ (underscore)

Click **Next**.

7. The verifying installation parameters window is displayed (Figure 3-34). Once the parameters have been verified, the installation summary window is displayed.



Please read the summary information below.

IBM Tivoli Dynamic Workload Console will be installed in the following location:

/opt/IBM/ISC

with the following features:

Integrated Solution Console Runtime
IBM Tivoli Dynamic Workload Console Module

for a total size:

618 MB

Figure 3-34 The verifying installation parameters window

Click **Install**.

8. When the installation progress window is displayed, you can stop the installation at any time by clicking **Cancel**. The installation will complete its current installation step, then suspend the installation. You will be asked if you want to cancel the installation. If you choose **Yes**, the current installation is cancelled and the installation summary window is displayed showing the reason for the cancellation.

9. When the installation has successfully completed, an installation summary screen is displayed showing the operations that have been performed during the installation.

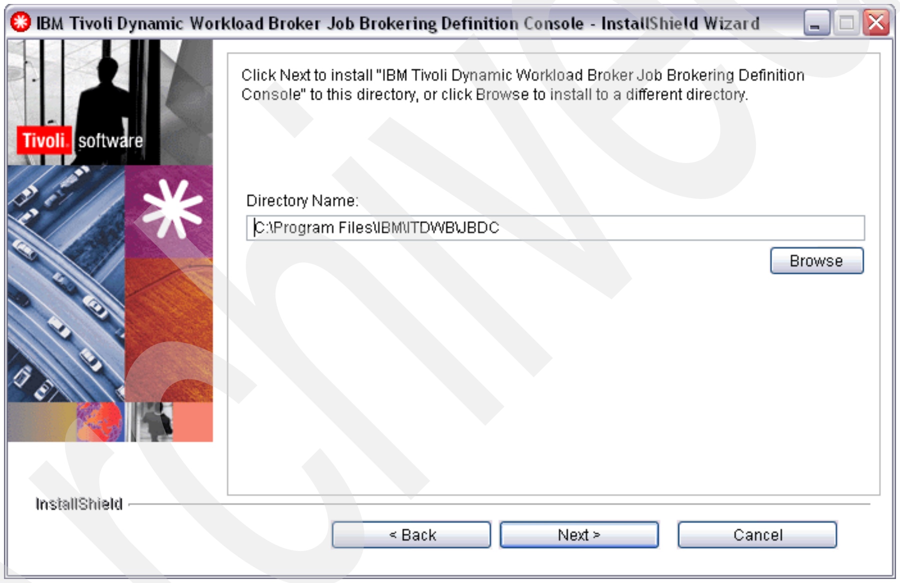
   When the installation has completed successfully, and the installation summary window is displayed, take note of the URLs displayed on this panel. You will need this information to connect to the ISC later. Click **Next**. The installation completed window is displayed. Click **Finish** to close the installer.

3.3.4 Installing the Tivoli Dynamic Workload Broker Job Brokering Definition Console

To install:

1. Via the launchpad, click **Install the IBM Tivoli Dynamic Workload Broker Job Brokering Definition Console**. Select the language you want the wizard to display during the install, then click **OK**. The introduction window is displayed. Click **Next** to continue with the installation. The license agreement is displayed. You will then be prompted to read and accept the license agreement. You also have the choice to print out the license agreement. To continue, check **I accept both the IBM and non-IBM terms**, then click **Next**.

   The installation directory window is displayed, as seen in Figure 3-35.


IBM Tivoli Dynamic Workload Broker Job Brokering Definition Console - InstallShield Wizard

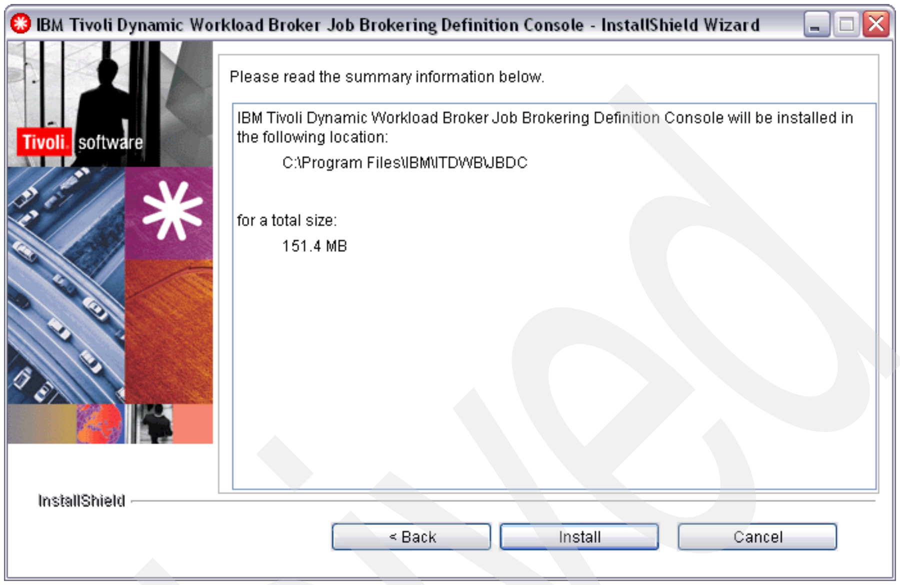
Click Next to install "IBM Tivoli Dynamic Workload Broker Job Brokering Definition Console" to this directory, or click Browse to install to a different directory.

Directory Name:
C:\Program Files\IBM\TDWB\JBDC

Figure 3-35 The installation directory window

2. You can either keep the default as listed, or click **Browse** to select a directory or drive of your choice. Click **Next** to accept the directory shown.

The installation summary window (Figure 3-36) is displayed showing the directory, features, and total size of the install.


IBM Tivoli Dynamic Workload Broker Job Brokering Definition Console - InstallShield Wizard

Please read the summary information below.

IBM Tivoli Dynamic Workload Broker Job Brokering Definition Console will be installed in the following location:

C:\Program Files\IBM\TDWB\JBDC

for a total size:

151.4 MB

Figure 3-36 The installation summary window

3. Click **Install**. IBM Tivoli Dynamic Workload Broker Job Brokering Definition Console will now be installed. When the installation progress window is displayed, you can stop the installation at any time by clicking **Cancel**. The installation will complete its current installation step, then suspend the installation. You will be asked if you want to cancel the installation. If you choose **Yes**, the current installation is cancelled and the installation summary window is displayed showing the reason for the cancellation.
4. When the installation has completed successfully, and the installation summary window is displayed, click **Next**. The installation completed window is displayed. Click **Finish** to close the installer.

3.3.5 Installing the IBM Tivoli Dynamic Workload Broker agent

Use the following procedure to install the Tivoli Dynamic Workload Broker agent component using the installation wizard:

1. Via the launchpad, click **Install the IBM Tivoli Dynamic Workload Broker Agent**. Select the language you want the wizard to display during the install, then click **OK**. The introduction window is displayed. Click **Next** to continue with the installation. The license agreement is displayed. You will then be

prompted to read and accept the license agreement. You also have the choice to print out the license agreement. To continue, check **I accept both the IBM and non-IBM terms**, then click **Next**.

2. The installation directory window is displayed.

   You can either keep the default directory as listed, or click **Browse** to select a directory or drive of your choice. Click **Next** to accept the directory shown. The Disconnected Mode window is displayed.

3. The installation requires a connection to the Tivoli Agent Manager and also to the Tivoli Dynamic Workload Broker server to be able to complete the deployment.

   If you are installing in a disconnected environment, you can check the **Install in Disconnected Mode** check box. This lets you install the agent without any connection checks being made. However, you will need to restart the agent when the computer is connected to the environment.

4. The Tivoli Common Agent Information window is displayed.

   Specify the label for the common agent and also the ports that the common agent will use, or accept the default values. Click **Next**.

5. The Windows User Information window is displayed.

   Specify the local system account, or check the **Specify User Account** check box to specify a specific user ID. Click **Next**.

6. The Agent Manager Information window is displayed. Specify the following information:

   – Agent Manager host name

     The fully qualified host name or the IP address of the Agent Manager server. This host name is used by the agents and the Tivoli Dynamic Workload Broker server to connect to the Tivoli Agent Manager.

   – Agent Manager Registration Port

     The port number for registration. The default is 9511. This port uses server-side authentication.

   – Agent Manager Public Port

     The port number for public communication, including the alternate port for the agent recovery service. The default is 9513.

   – Agent Manager Agent Registration Password

     The agent registration password. This password is presented by a common agent when it requests registration with the Agent Manager. This password also locks the agentTrust.jks truststore file, which contains the signer certificate for the Agent Manager.

- Agent Manager Context Root

  The context root of the application server. The default value is /AgentMgr. The context root is part of the URL that common agents and resource managers use to connect to the Agent Manager. For example, the context root is the underlined part of the following URL:

  ```
  http://bentley.tivlab.austin.ibm.com:9513/AgentMgr
  ```

  Click **Next**.

7. The Tivoli Dynamic Workload Broker server Information window is displayed. Specify the Tivoli Dynamic Workload Broker server host name and the port that the server will use, or accept the default values. Click **Next**.
8. The installation summary window is displayed showing the installation directory and the required disk space.

   Click **Install**. The Tivoli Dynamic Workload Broker agent will now be installed. When the installation progress window is displayed, you can stop the installation at any time by clicking **Cancel**. The installation will complete its current installation step, then suspend the installation. You will be asked if you want to cancel the installation. If you choose **Yes**, the current installation is cancelled and the installation summary window is displayed showing the reason for the cancellation.

   When the installation complete window is displayed showing the operations that have been performed during the installation, click **Next**. The installation completed window is displayed. Click **Finish** to close the installer.

3.4 Uninstallation

Using the Add/Remove Programs feature on Windows, uninstall the Tivoli Dynamic Workload Broker components. For AIX, UNIX, and any other instance where the software was installed with alternate methods, you need to use the commands or utilities to uninstall the components.

If, for any reason, the uninstaller fails, the <installation_directory>_uninst.resume directory is created. Before running the uninstaller again, you should rename the _uninst.resume directory to _uninst and then run the uninstaller with the -resume option. If the uninstall is still failing, then follow the procedure for manual uninstallation.

Manual uninstall of the Agent Manager

Use the following procedure for a manual uninstall of the Agent Manager:

1. Connect to AgentManager WebSphere Admin Console (http://<am_machine>:9060/admin/).
2. From **Applications** → **Enterprise Applications** select **AgentManager** and **AgentRecoveryService**, then click **Remove File**.
3. From **Resources** → **JDBC™ Providers** → **AgentJDBCProvider** → **Data sources** → **AgentRegistry** → **J2EE Connector Architecture (J2C) authentication data entries** select **AgentRegistryDBAuth**, then click **Delete**.
4. From **Resources** → **JDBC Providers** → **AgentJDBCProvider** → **Data sources** select **AgentRegistry**, then click **Delete**.
5. From **Resources** → **JDBC Providers** select **AgentJDBCProvider**, then click **Delete**.
6. From **Environment** → **Virtual Hosts** select **AgentManagerHost**, then click **Delete**.
7. At the top of the page, from the Messages box, click **Save** to apply the changes made.
8. Restart WebSphere.
9. Remove the Agent Manager installation directory (see default installation directories).
10. (Optional) Drop the Agent Manager database.
11. Remove all entries related to the AgentManager from vpd.properties and vpd.script.

Note: If the Agent Manager is the only component installed, and there are no other references in the file, the vpd.properties file can be removed all together.

Manual uninstall of the Tivoli Dynamic Workload Broker server and Tivoli Workload Scheduler agent

Use the following procedure for a manual uninstall of the Tivoli Dynamic Workload Broker server and Tivoli Workload Scheduler agent:

1. Connect to the Tivoli Dynamic Workload Broker WebSphere Administration Console (`http://<itdwb>:9060/admin/`).
2. From **Applications** → **Enterprise Applications** select **ITDWB** and **TWSAgent**, then click **Remove File**.

3. From **Resources** → **JDBC Providers** → **ITDWBProvider** → **Data sources** → **ITDWBDataSource** → **J2EE Connector Architecture (J2C) authentication data entries** select **ITDWBDB2Access**, then click **Delete**.
4. From **Resources** → **JDBC Providers** → **ITDWBProvider** → **Data sources** select **ITDWBDataSource**, then click **Delete**.
5. From **Resources** → **JDBC Providers** → select **ITDWBProvider**, then click **Delete**.
6. From **Resources** → **Asynchronous beans** → **Work Managers** select **JobDispatcherWorkManager** and **ResourceAdvisorWorkManager**, then click **Delete**.
7. From **Environment** → **Virtual Hosts** select **ITDWBHost**, then click **Delete**.
8. At the top of the page, from the Messages box, click **Save** to apply the changes made.
9. Restart the WebSphere Application Server.
10. Remove the Tivoli Dynamic Workload Broker installation directory (see default directories in Table 3-3 on page 92).
11. (Optional) Drop the TDWB database.
12. Remove the /root/InstallShield/Universal/IBMVPD directory.

> **Note:** If the Agent Manager and the Tivoli Dynamic Workload Broker server are installed on a WAS profile that is dedicated to them, you can avoid the above steps by deleting the WebSphere profile used for the installation, then drop the databases as needed.

4

Working with Tivoli Dynamic Workload Broker

This chapter discusses the concepts and terminology used in Tivoli Dynamic Workload Broker for running jobs. Tivoli Dynamic Workload Broker manages computers and assigns jobs to run on the computers that it manages. It monitors both the jobs and the computers. Computers managed by Tivoli Dynamic Workload Broker can be added, removed, or changed without necessarily changing the jobs.

This chapter contains the following sections:

4.1 Computers

The *Tivoli Dynamic Workload Broker server* is the product component that manages the computers that run jobs. This component:

- Assigns resources to jobs
- Stores resources and job definitions in a database
- Maintains computer configuration information in a database

Computers are physical machines and are called *workstations*. The *Tivoli Dynamic Workload Broker Workload agent* is installed on each computer managed by the *Tivoli Dynamic Workload Broker server.*

Workload on the Tivoli Dynamic Workload Broker server should be minimized so that it can monitor the computers and jobs without interference from the jobs themselves. For this reason we do not recommend installation of the Tivoli Dynamic Workload Broker Workload agent on the Tivoli Dynamic Workload Broker server machine.

4.1.1 Resources

Computers are defined to the Tivoli Dynamic Workload Broker server as *resources*. A resource is a physical computer or a *logical resource*. A logical resource is an entity that can be associated with one or more computers to represent applications, groups, licenses, servers, and other traits not associated with a specific computer. Computers can be grouped together in a *resource group*. A resource group can contain physical computers, logical resources, or both.

Tivoli Dynamic Workload Broker assigns jobs to resources or resource groups. This is known as the virtualization of resources and is the main idea of Tivoli Dynamic Workload Broker. If a computer is removed, then you can change the resources that reference the removed computer. In that way the job does not need to be changed if the resource is updated to reference a different computer upon which the job may run.

Note: A job definition that uses a resource group containing both computers and logical resources will require two Job Submission Description Language (JSDL) files (see 4.2.1, "Job definitions" on page 143 for a discussion of job definitions and JSDL). It is simpler to have resource groups that contain only computers or only logical resources, not both.

4.1.2 Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent plug-in

The *Tivoli Dynamic Workload Broker Tivoli Workload Scheduler agent* (also referred to as the TWS Agent) is a bridge between the Tivoli Dynamic Workload Broker server and a Tivoli Workload Scheduler domain manager. The TWS Agent is installed on the Tivoli Dynamic Workload Broker server machine. This machine is configured in Tivoli Workload Scheduler as a standard agent but does not have all of the functions of a standard agent. The TWS Agent is not a Tivoli Dynamic Workload Broker Workload Agent and is not one of the computers managed by Tivoli Dynamic Workload Broker for job submission.

The TWS Agent is a plug-in for the Tivoli Dynamic Workload Broker server that may be installed after the Tivoli Dynamic Workload Broker server installation without uninstalling the Tivoli Dynamic Workload Broker server.

4.2 Working with jobs

A job is an executable file or a script that is assigned to a resource by the Tivoli Dynamic Workload Broker server. The resource defines the Tivoli Dynamic Workload Broker managed computers that may be used to run the job. Tivoli Dynamic Workload Broker provides the status of the jobs and the managed computers.

4.2.1 Job definitions

Each job is created as a *job definition*. Job definitions are stored in the Tivoli Dynamic Workload Broker Job Repository DB2 database. A job definition is written using Job Submission Description Language (JSDL) statements. This is sometimes also called Job Submission Definition Language in the IBM manuals.

A job definition written in JSDL describes the job requirements for submission to resources. The JSDL language contains a vocabulary and normative XML schema that facilitate the expression of those requirements as a set of XML elements. Knowledge of XML is helpful, but not essential, for working with job definitions. Job definitions are created in files with the extension of *jsdl*.

Example 4-1 is an example of a job definition with:

- The name is testjob1.
- The executable is a file test1script.ksh that must exist on a Tivoli Dynamic Workload Broker managed computer in the directory /tmp.

There is no resource specified so the Tivoli Dynamic Workload Broker server will assign an available resource. There may be different versions of the script /tmp/test1script.ksh on different computers managed by Tivoli Dynamic Workload Broker. This could result in different job behavior for each job submission depending upon the computer selected by Tivoli Dynamic Workload Broker to run the job.

Example 4-1 Job definition jsdl file referencing an executable file

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
description=" Test Job 1" name="testjob1">
<jsdl:annotation>Job definition created within JBDC</jsdl:annotation>
<jsdl:application name="executable">
<jsdle:executable path="/tmp/test1script.ksh"/>
</jsdl:application>
</jsdl:jobDefinition>
```

Example 4-2 is an example of a job definition with:

- The name is “testjob2”.
- The executable is a script that is contained in the job definition.
- The resource to use is the computer “gridnode0135”.

The script to run is part of the job definition. You can change the resource and the script will remain the same because it is part of the job definition.

Example 4-2 Job definition jsdl file containing an executable script

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
description="Test Job 2" name="testjob2">

<jsdl:application name="executable">

<jsdle:executable>
<jsdle:script>#!/bin/ksh
```

```
echo " test job 2: Hello World !!!"
</jsdle:script>
</jsdle:executable>

</jsdl:application>

<jsdl:resources>
<jsdl:candidateHosts>
<jsdl:hostName>gridnode0135</jsdl:hostName>
</jsdl:candidateHosts>
</jsdl:resources>

</jsdl:jobDefinition>
```

Example 4-3 is an example of a job definition where:

- The name is “testjob3”.
- The executable is a script that is contained in the job definition.
- The resource to use is the group “computers”.

In this example only physical computers are in the group “computers”. Unpredictable results will occur if the group “computers” contains both physical computers and logical resources.

Example 4-3 Job definition jsdl file with a computer resource group

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
description="resource group - only computers" name="testjob3">
<jsdl:application name="executable">
<jsdle:executable>
<jsdle:script>
#!/bin/ksh
echo "running test3 script"
</jsdle:script>
</jsdle:executable>
</jsdl:application>
<jsdl:resources>
<jsdl:group name="computers"/>
</jsdl:resources>
</jsdl:jobDefinition>
```

Example 4-4 is an example of a job definition where:

- The name is “testjob3”.
- The executable is a script that is contained in the job definition.
- The resource to use is the group “logical”.

In this example only logical resources are in the group “logical”. Unpredictable results will occur if the group “logical” contains both physical computers and logical resources.

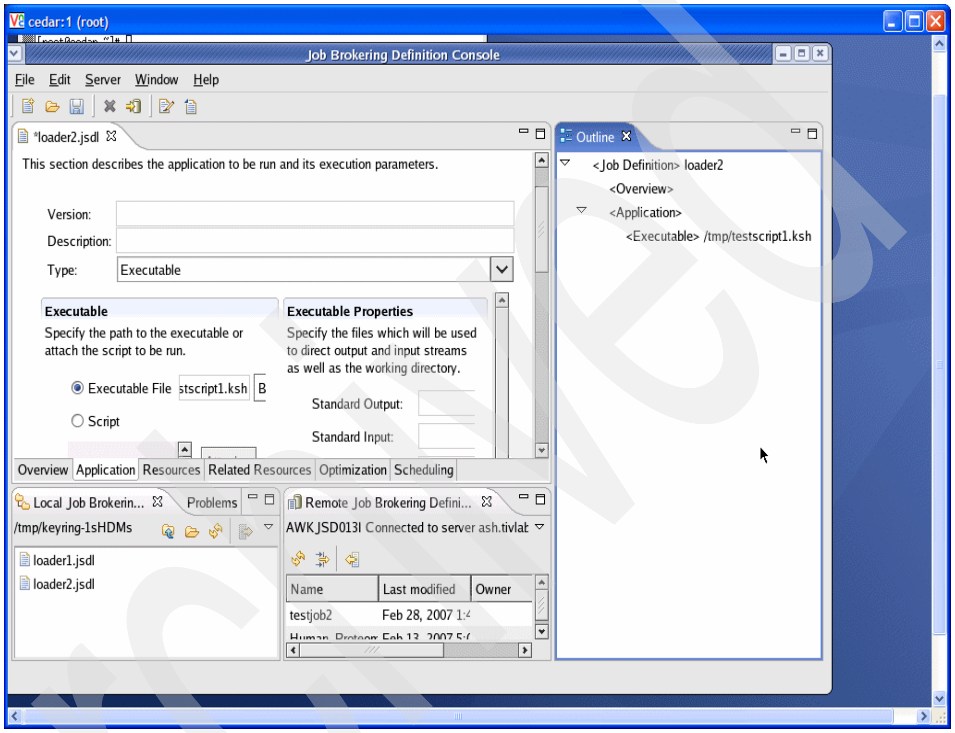
Note: The XML for use of a group with logical resources is more complicated than that for use of a group with physical computers. Be sure to change the XML statements accordingly when switching between a group with physical computers to a group with logical resources for a job definition.

Example 4-4 Job definition jsdl file with a logical resource group

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
description="resource group - only logical resources" name="testjob4">
<jsdl:application name="executable">
<jsdle:executable>
<jsdle:script>
#!/bin/ksh
echo "running testjob4"
</jsdle:script>
</jsdle:executable>
</jsdl:application>
<jsdl:resources>
<jsdl:relationship target="logicalresources" type="AssociatesWith"/>
</jsdl:resources>
<jsdl:relatedResources id="logicalresources" type="LogicalResource">
<jsdl:group name="logical"/>
</jsdl:relatedResources>
</jsdl:jobDefinition>
```

4.2.2 JBDC, Web Console, and command-line interface

The *Tivoli Dynamic Workload Broker Job Brokering Definition Console (JBDC)* is an application that communicates with the Tivoli Dynamic Workload Broker server and provides a graphical user interface (GUI) for creating job definitions. See Figure 4-1.


cedar:1 (root)
Job Brokering Definition Console
File Edit Server Window Help
*loader2.jsdl
Outline
Overview Application Resources Related Resources Optimization Scheduling
Local Job Brokerin...
Problems
Remote Job Brokering Defini...

This section describes the application to be run and its execution parameters.

Version:
Description:
Type: Executable

Executable
Specify the path to the executable or attach the script to be run.
(●) Executable File stscript1.ksh
( ) Script

Executable Properties
Specify the files which will be used to direct output and input streams as well as the working directory.
Standard Output:
Standard Input:

<Job Definition> loader2
<Overview>
<Application>
<Executable> /tmp/testscript1.ksh

/tmp/keyring-1sHDMs
loader1.jsdl
loader2.jsdl

AWKJSD013I Connected to server ash.tivlab

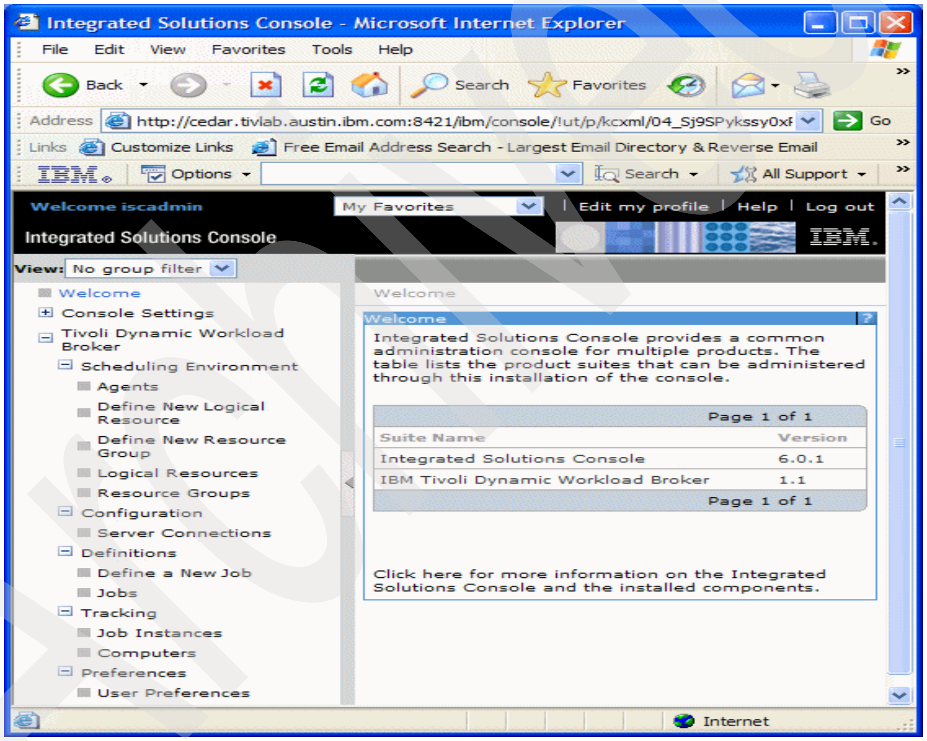
Name	Last modified	Owner
testjob2	Feb 28, 2007 1:4	
Human_Protoom	Feb 13, 2007 5:(	

Figure 4-1 Tivoli Dynamic Workload Broker JBDC graphical user interface

The JBDC can be installed on a user workstation that is separate from the Tivoli Dynamic Workload Broker server and the Tivoli Dynamic Workload Broker agent installed on each computer managed by the Tivoli Dynamic Workload Broker server. You can create new job definitions and upload them to the Tivoli Dynamic Workload Broker server or download/change/upload existing job definitions on the Tivoli Dynamic Workload Broker server. A job definition must be stored in the DB2 database by the Tivoli Dynamic Workload Broker server before it can be run.

The JBDC GUI does not provide all of the JSDL elements that may be needed for a job definition. One strategy to follow is to use the JBDC GUI to create the job definition and then use the *Tivoli Dynamic Workload Broker Web Console* to edit the job definition in order to add additional elements or make changes.

The Tivoli Dynamic Workload Broker Web Console, as shown in Figure 4-2, runs on the *Integrated Solutions Console (ISC)*. The Integrated Solutions Console will be installed automatically if it is not already installed as part of the installation of the Tivoli Dynamic Workload Broker Web Console. You can use the Tivoli Dynamic Workload Broker Web Console to create or update a job definition from the Tivoli Dynamic Workload Broker Web Console editor using JSDL statements. This means that you will need to know the JSDL schema in order to create a job definition or make significant changes to an existing job definition. However, simple changes such as changing an executable file name may be possible without knowledge of the JSDL schema.


Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://cedar.tivlab.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/04_Sj9SPykssy0xF Go
Links Customize Links Free Email Address Search - Largest Email Directory & Reverse Email
Options Search All Support
Welcome iscadmin
My Favorites
Edit my profile | Help | Log out
Integrated Solutions Console
View: No group filter
Welcome
Console Settings
Tivoli Dynamic Workload Broker
Scheduling Environment
Agents
Define New Logical Resource
Define New Resource Group
Logical Resources
Resource Groups
Configuration
Server Connections
Definitions
Define a New Job
Jobs
Tracking
Job Instances
Computers
Preferences
User Preferences
Welcome
Internet

Welcome

Integrated Solutions Console provides a common administration console for multiple products. The table lists the product suites that can be administered through this installation of the console.

Page 1 of 1

Suite Name	Version
Integrated Solutions Console	6.0.1
IBM Tivoli Dynamic Workload Broker	1.1

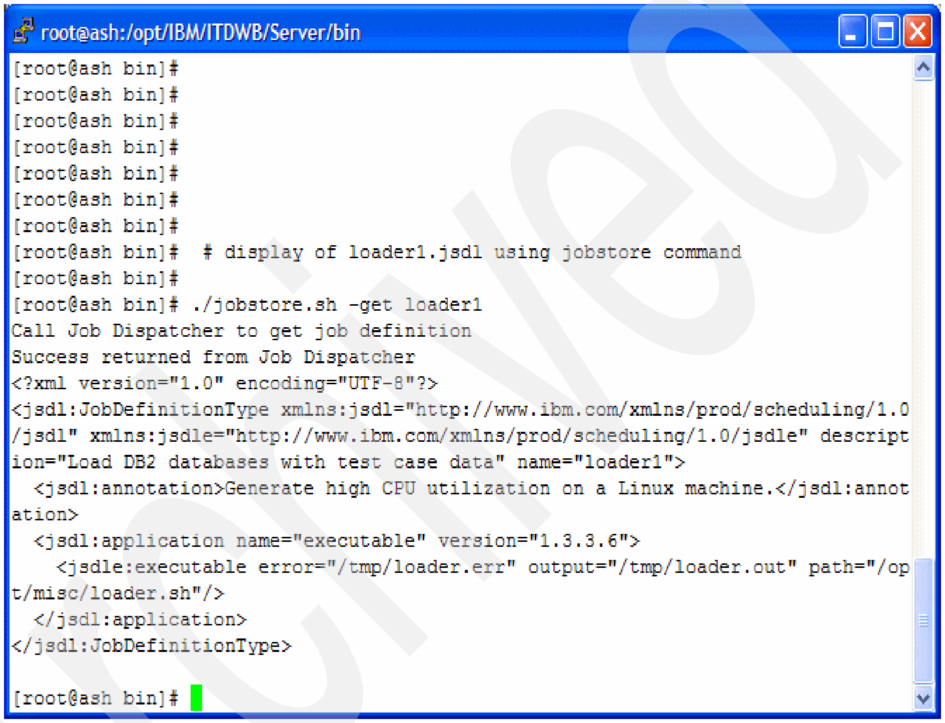
Page 1 of 1

Click here for more information on the Integrated Solutions Console and the installed components.

Figure 4-2 Tivoli Dynamic Workload Broker Web Console

The Tivoli Dynamic Workload Broker command-line interface runs on the Tivoli Dynamic Workload Broker server. The jobstore script can be used to create, display, update, and delete Tivoli Dynamic Workload Broker job definitions. See Figure 4-3.

Note: You need to run the tdwb_env script before using the Tivoli Dynamic Workload Broker CLI. This script is found in the <install directory>/bin.



```
[root@ash bin]#
[root@ash bin]#
[root@ash bin]#
[root@ash bin]#
[root@ash bin]#
[root@ash bin]#
[root@ash bin]#
[root@ash bin]#  # display of loader1.jsdl using jobstore command
[root@ash bin]#
[root@ash bin]# ./jobstore.sh -get loader1
Call Job Dispatcher to get job definition
Success returned from Job Dispatcher
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinitionType xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0
/jsdl" xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" descript
ion="Load DB2 databases with test case data" name="loader1">
  <jsdl:annotation>Generate high CPU utilization on a Linux machine.</jsdl:annot
ation>
  <jsdl:application name="executable" version="1.3.3.6">
    <jsdle:executable error="/tmp/loader.err" output="/tmp/loader.out" path="/op
t/misc/loader.sh"/>
  </jsdl:application>
</jsdl:JobDefinitionType>

[root@ash bin]#
```

Figure 4-3 Tivoli Dynamic Workload Broker CLI jobstore -get to display job definition

4.2.3 Job submission

Jobs may be submitted by:

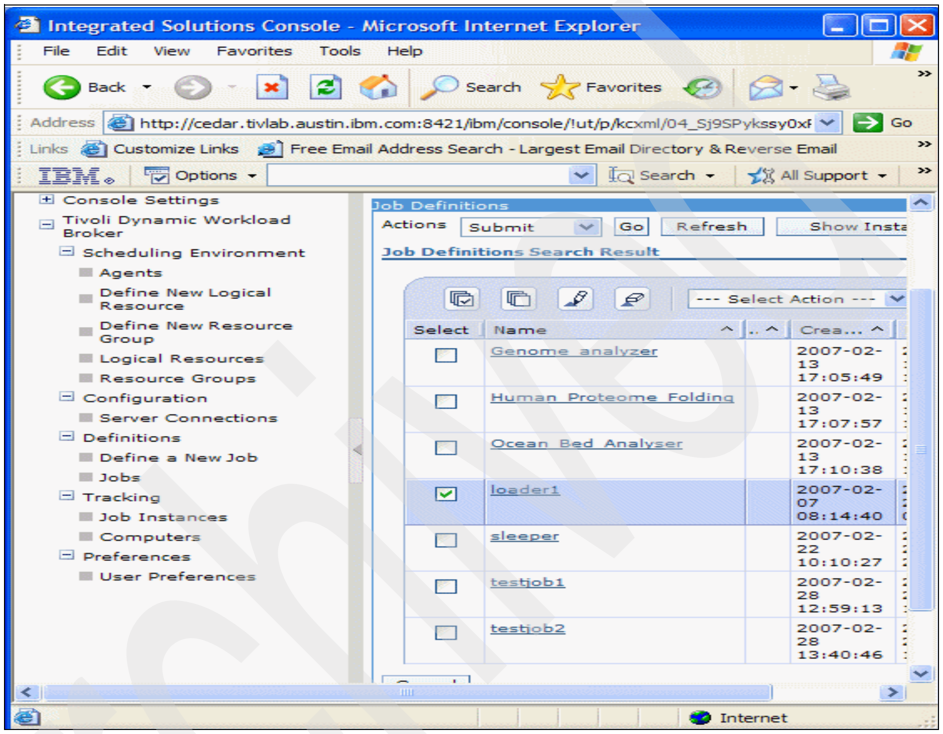
- Tivoli Workload Scheduler job streams
- Tivoli Dynamic Workload Broker Web Console
- Tivoli Dynamic Workload Broker CLI (command-line interface)

Tivoli Workload Scheduler job streams

Tivoli Dynamic Workload Broker jobs can be submitted in Tivoli Workload Scheduler as job streams. This includes the *ad hoc* submission of a Tivoli Workload Scheduler job definition. A Tivoli Workload Scheduler job definition must run on the Tivoli Dynamic Workload Broker server upon which the TWS Agent is installed. The TWS Agent is the communications bridge for the Tivoli Dynamic Workload Broker job referenced in a Tivoli Workload Scheduler job definition. It is important to note that Tivoli Dynamic Workload Broker does not have the concept of *job stream* as found in Tivoli Workload Scheduler. There is a combination of strengths of Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler. Tivoli Workload Scheduler controls the sequence and dependency resolution within the job streams that contain Tivoli Workload Scheduler job definitions that reference Tivoli Dynamic Workload Broker job definitions. Tivoli Dynamic Workload Broker does the load balancing of Tivoli Dynamic Workload Broker jobs when submitted by Tivoli Workload Scheduler.

Tivoli Dynamic Workload Broker Web Console

The Tivoli Dynamic Workload Broker Web Console has a Tivoli Dynamic Workload Broker job submission function, as shown in Figure 4-4. This allows you to submit Tivoli Dynamic Workload Broker jobs directly to the Tivoli Dynamic Workload Broker server in an independent fashion.



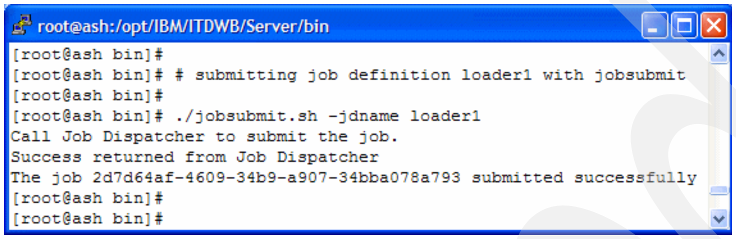
Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://cedar.tivlab.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/04_Sj9SPykssy0xf Go
Links Customize Links Free Email Address Search - Largest Email Directory & Reverse Email
IBM Options Search All Support
Console Settings
Tivoli Dynamic Workload Broker
Scheduling Environment
Agents
Define New Logical Resource
Define New Resource Group
Logical Resources
Resource Groups
Configuration
Server Connections
Definitions
Define a New Job
Jobs
Tracking
Job Instances
Computers
Preferences
User Preferences
Job Definitions
Actions Submit Go Refresh Show Insta
Job Definitions Search Result
--- Select Action ---
Internet

Select	Name	..	Crea...
☐	Genome analyzer		2007-02-13 17:05:49
☐	Human Proteome Folding		2007-02-13 17:07:57
☐	Ocean Bed Analyser		2007-02-13 17:10:38
✓	loader1		2007-02-07 08:14:40
☐	sleeper		2007-02-22 10:10:27
☐	testjob1		2007-02-28 12:59:13
☐	testjob2		2007-02-28 13:40:46

Figure 4-4 Tivoli Dynamic Workload Broker Web Console job definition Submit function

Tivoli Dynamic Workload Broker command-line interface

The Tivoli Dynamic Workload Broker CLI jobsubmit script can be used to submit a Tivoli Dynamic Workload Broker job. See Figure 4-5. This allows you to use a CLI tool instead of a graphical user interface to submit Tivoli Dynamic Workload Broker jobs in an independent fashion.



```
[root@ash bin]#
[root@ash bin]# # submitting job definition loader1 with jobsubmit
[root@ash bin]#
[root@ash bin]# ./jobsubmit.sh -jdname loader1
Call Job Dispatcher to submit the job.
Success returned from Job Dispatcher
The job 2d7d64af-4609-34b9-a907-34bba078a793 submitted successfully
[root@ash bin]#
[root@ash bin]#
```

Figure 4-5 jobsubmit -jdname to submit job definition loader1

4.2.4 Credentials for job definitions

The *credential element* of the Job Scheduling Definition Language specifies the security credential for running the command specified in the job definition. This element can be created in the Job Brokering Definition Console using the Application tab.

The related elements are:

- userName - user name to be used when running the command
- groupName - group name to be used when running the command
- password - password to be used when running the command (This is not encrypted in the job definition file.)

Note: For Windows the user ID and password are ignored in Tivoli Dynamic Workload Broker V1.1.

You can also define the user and password for J2EE jobs. You can use the Job Brokering Definition Console Application tab for J2EE and specify the J2EE credentials.

4.2.5 Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker job definitions

The concept of a job definition in Tivoli Workload Scheduler and in Tivoli Dynamic Workload Broker is similar. In both products, a job definition defines what will run (the job) and where it will run (workstation in Tivoli Workload Scheduler, resource in Tivoli Dynamic Workload Broker).

Ther Tivoli Workload Scheduler agent plug-in must be installed on the Tivoli Dynamic Workload Broker server in order to provide a bridge from Tivoli Workload Scheduler to Tivoli Dynamic Workload Broker for running jobs. In addition, the Tivoli Dynamic Workload Broker server computer is configured as a Tivoli Workload Scheduler Standard Agent in Tivoli Workload Scheduler. However, only Tivoli Workload Scheduler job definitions with the Task field containing a Tivoli Dynamic Workload Broker job definition reference will run on the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent.

Note: The Tivoli Dynamic Workload Broker job definition referenced in a Tivoli Workload Scheduler job definition must reference a logical resource.

The Tivoli Dynamic Workload Broker Tivoli Workload Scheduler agent is not a Tivoli Dynamic Workload Broker Workload Agent. When the Tivoli Workload Scheduler job arrives, the Tivoli Dynamic Workload Broker server retrieves the Tivoli Dynamic Workload Broker job definition referenced in the Tivoli Workload Scheduler job definition Task field, assigns it a Tivoli Dynamic Workload Broker resource, and runs it on a Tivoli Dynamic Workload Broker managed computer.

Tivoli Workload Scheduler sees the Tivoli Workload Scheduler job as running on the Tivoli Dynamic Workload Broker server. The Tivoli Dynamic Workload Broker server sees the Tivoli Dynamic Workload Broker job running on one of the Tivoli Dynamic Workload Broker managed computers. The Tivoli Dynamic Workload Broker server reports the status and job output back to Tivoli Workload Scheduler through the TWS Agent plug-in on the Tivoli Dynamic Workload Broker server.

It is important to note that Tivoli Workload Scheduler cannot directly specify which Tivoli Dynamic Workload Broker managed computer will be used to run the Tivoli Dynamic Workload Broker job. However, it is possible to create a Tivoli Dynamic Workload Broker job definition that contains a job variable whose value is a resource. In this situation you can specify the value for the Tivoli Dynamic Workload Broker job variable in the Tivoli Workload Scheduler job definition Task field. (See 4.3.1, “Job variables” on page 155.)

Tivoli Workload Scheduler is a job scheduling product and Tivoli Dynamic Workload Broker is a workload management product. Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker work together to provide scheduled jobs in a workload managed environment.

4.2.6 Job affinity

An affinity relationship is established between two or more jobs when you want them to run on the same resource. This is useful for situations in which the results of one job are needed for the next job.

Affinity between two or more jobs can be defined using:

- Tivoli Dynamic Workload Broker Web Console
- Tivoli Dynamic Workload Broker command-line interface (`jobsubmit` command)
- Tivoli Workload Scheduler

Tivoli Dynamic Workload Broker Web Console

From the Tivoli Dynamic Workload Broker Web Console you select **Submit** rather than **Submit** to get to the wizard that facilitates affine jobs. You then specify the job name or the alias name for the affine job. The job definition that you submit will attempt to run on the same machine as the affine job.

Submit is also used for specifying a job alias, values for job variables, and a target resource.

Note: If a job definition specifies a candidateHost that is different from the computer used for the affine job, then the job will fail with *Resource allocation failed*.

Tivoli Dynamic Workload Broker jobsubmit command

The jobsubmit script requires a job ID or an alias name for the affine job. For this reason it is a good idea to always specify an alias name for an affine job, as that is usually easier to remember than the job ID. The job definition that you submit will attempt to run on the same machine as the affine job.

Tivoli Workload Scheduler

To define Tivoli Dynamic Workload Broker affinity between two or more jobs, specify the affinity relationship in the Task String section using either the Tivoli Dynamic Workload Broker job ID or the job alias:

```
“<jobName> -affinity jobid=<jobid>”
“<jobName> -affinity alias=<alias>”
```

To define the Tivoli Workload Scheduler affinity between tow or more jobs in the same job stream, specify the affinity relationship in the Task String section:

```
“<jobName> -twsAffinity jobname=<twsJobName>”
```

This affinity definition overrides the affinity with the job ID and the affinity with the job alias.

4.3 Using variables in job definitions

There are two types of variables in a job definition:

- Job variables (defined by the *variables* JSDL element)
- Environment variables (defined by the *environment* JSDL element)

Use of variables adds flexibility to the job definitions.

4.3.1 Job variables

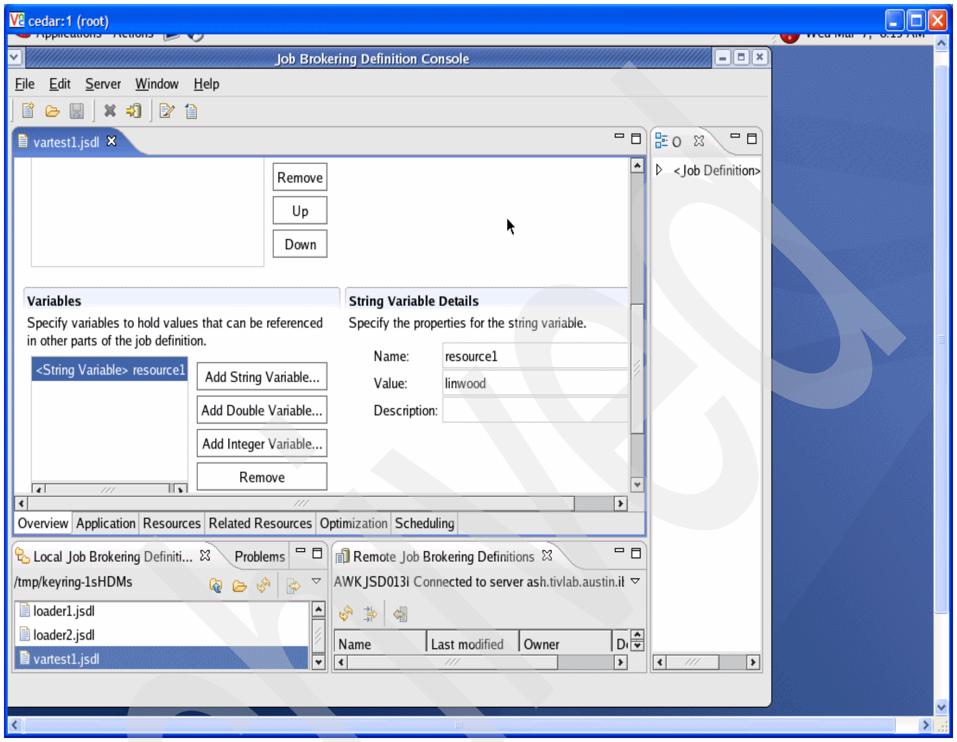
A job definition can be created with job variables that will be assigned values from the job submission. In this way you can have one job definition that is used for several different situations by specifying the desired value for a job variable.

Note: Job variables are used in the JSDL statements of a job definition and not in the script that is to be run.

There are three types of job variables in Tivoli Dynamic Workload Broker:

- String (string of characters)
- Double (decimal value)
- Integer (integer value, no decimal point)

Job variables are defined in a job definition as the JSDL variables element. Job variables can be defined in the Tivoli Dynamic Workload Broker JBDC, as shown in Figure 4-6.


cedar:1 (root)
Job Brokering Definition Console
File Edit Server Window Help
vartest1.jsdl
Remove
Up
Down
<Job Definition>
Add String Variable...
Add Double Variable...
Add Integer Variable...
Remove
Overview Application Resources Related Resources Optimization Scheduling
Local Job Brokering Definiti...
Problems
Remote Job Brokering Definitions
Name Last modified Owner

Variables

Specify variables to hold values that can be referenced in other parts of the job definition.

<String Variable> resource1

String Variable Details

Specify the properties for the string variable.

Name: resource1

Value: linwood

Description:

/tmp/keyring-1sHDMs

loader1.jsdl
loader2.jsdl
vartest1.jsdl

AWKJSD013I Connected to server ash.tivlab.austin.il

Figure 4-6 Tivoli Dynamic Workload Broker JBDC GUI for creating job variable resource1

This job variable *resource1* has the value of one of the computers managed by Tivoli Dynamic Workload Broker - *linwood*.

The *Resources* element in the job definition is created using the variable resource1, as in Figure 4-7.

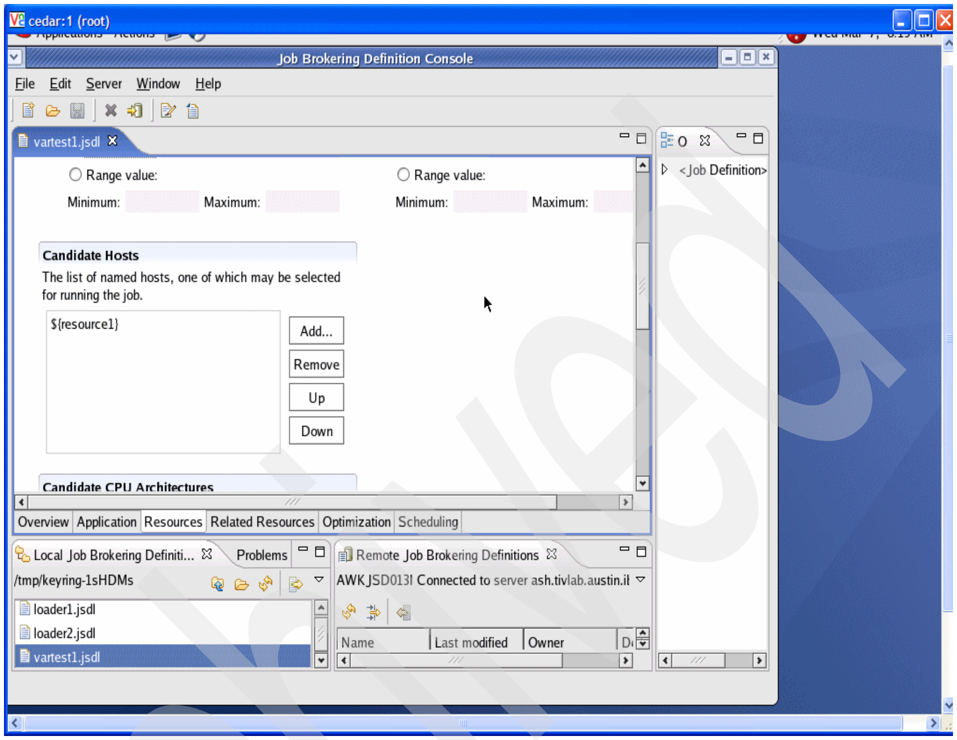


Figure 4-7   Resources element will reference the resource1 job variable

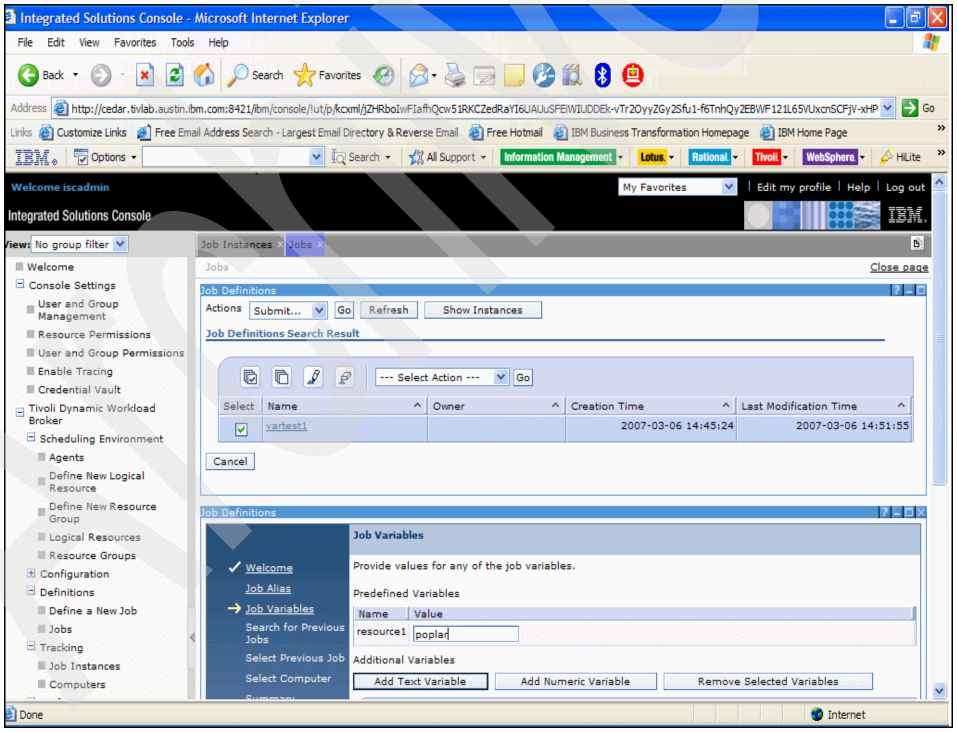
Job variables are referenced in the XML starting with ${ and ending with }. For this example, Tivoli Dynamic Workload Broker assigns the resource *linwood* to the job *vartest1* unless the variable *resource1* is assigned a different value during job submission from the Tivoli Dynamic Workload Broker Web Console, from the jobsubmit.sh script (Tivoli Dynamic Workload Broker CLI), or from the Tivoli Workload Scheduler Task field for the related Tivoli Workload Scheduler job definition. See Example 4-5.

Example 4-5   Job definition vartest1 with Tivoli Dynamic Workload Broker job variable resource1

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
description="test job variables" name="vartest1">
  <jsdl:variables>
```

```
    <jsdl:stringVariable name="resource1">linwood</jsdl:stringVariable>
  </jsdl:variables>
  <jsdl:application name="executable">
    <jsdle:executable>
      <jsdle:script>#!/bin/ksh
echo "test vartest1"</jsdle:script>
    </jsdle:executable>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:candidateHosts>
      <jsdl:hostName>${resource1}</jsdl:hostName>
    </jsdl:candidateHosts>
  </jsdl:resources>
</jsdl:jobDefinition>
```

Figure 4-8 shows the use of the Tivoli Dynamic Workload Broker Web Console to change the value of variable resource1 as it exists in the job definition.


Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://cedar.tivlab.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/jZHRboIwFIafhQcw51RKCZedRaYI6UAUuSFEIWIUDDEk-vTr2OyyZGy2Sfu1-f6TnhQy2EBWF121L65VUxcnSCFjV-xHP
Welcome iscadmin
Integrated Solutions Console
My Favorites | Edit my profile | Help | Log out
View: No group filter
Welcome
Console Settings
User and Group Management
Resource Permissions
User and Group Permissions
Enable Tracing
Credential Vault
Tivoli Dynamic Workload Broker
Scheduling Environment
Agents
Define New Logical Resource
Define New Resource Group
Logical Resources
Resource Groups
Configuration
Definitions
Define a New Job
Jobs
Tracking
Job Instances
Computers
Job Instances
Jobs
Close page
Job Definitions
Actions Submit... Go Refresh Show Instances
Job Definitions Search Result
--- Select Action --- Go
Cancel
Job Definitions
Welcome
Job Alias
Job Variables
Search for Previous Jobs
Select Previous Job
Select Computer
Add Text Variable
Add Numeric Variable
Remove Selected Variables
Done
Internet

Select	Name	Owner	Creation Time	Last Modification Time
✓	vartest1		2007-03-06 14:45:24	2007-03-06 14:51:55

Job Variables

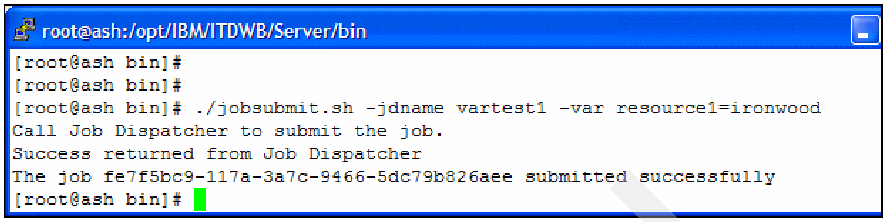
Provide values for any of the job variables.

Predefined Variables

Name	Value
resource1	poplar

Additional Variables

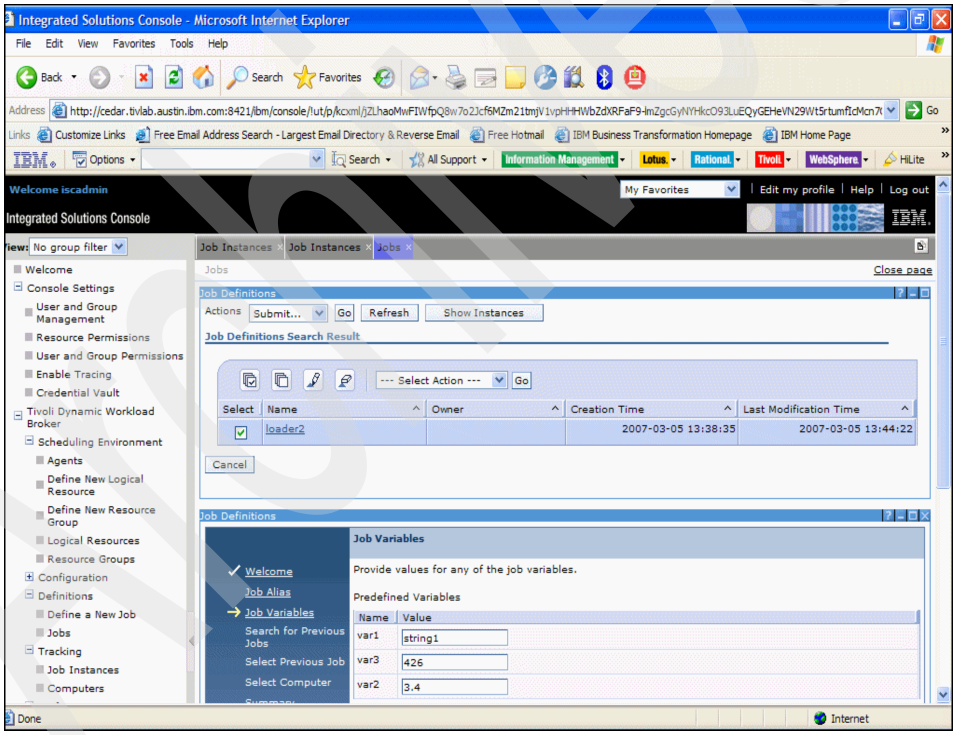
Figure 4-8 Tivoli Dynamic Workload Broker job variable resource1 will be set to name of the computer poplar



```
root@ash:/opt/IBM/ITDWB/Server/bin
[root@ash bin]#
[root@ash bin]#
[root@ash bin]# ./jobsubmit.sh -jdname vartest1 -var resource1=ironwood
Call Job Dispatcher to submit the job.
Success returned from Job Dispatcher
The job fe7f5bc9-117a-3a7c-9466-5dc79b826aee submitted successfully
[root@ash bin]#
```

Figure 4-9 Tivoli Dynamic Workload Broker CLI jobsubmit.sh script sets the job variable resource1 to the name of the computer ironwood

Note: Job variables cannot be directly used in the script that is stored as part of the job definition. See 4.3.3, “Indirect use of job variables in scripts” on page 161 for a technique to indirectly reference job variables.



Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://cedar.tivlab.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/jZLhaoMwFIWfpQ8w7o2Jcf6MZm21tmjV1vpHHHWbZdXRFaF9-lmZgcGyNYHkcO93LuEQyGEHeVN29Wt5rtumfIcMcn7
Welcome iscadmin
Integrated Solutions Console
My Favorites | Edit my profile | Help | Log out
View: No group filter
Welcome
Console Settings
User and Group Management
Resource Permissions
User and Group Permissions
Enable Tracing
Credential Vault
Tivoli Dynamic Workload Broker
Scheduling Environment
Agents
Define New Logical Resource
Define New Resource Group
Logical Resources
Resource Groups
Configuration
Definitions
Define a New Job
Jobs
Tracking
Job Instances
Computers
Job Instances | Job Instances | Jobs
Close page
Job Definitions
Actions Submit... Go Refresh Show Instances
Job Definitions Search Result
--- Select Action --- Go
Cancel
Welcome
Job Alias
Job Variables
Search for Previous Jobs
Select Previous Job
Select Computer
Done
Internet

Select	Name	Owner	Creation Time	Last Modification Time
✓	loader2		2007-03-05 13:38:35	2007-03-05 13:44:22

Job Variables

Provide values for any of the job variables.

Predefined Variables

Name	Value
var1	string1
var3	426
var2	3.4

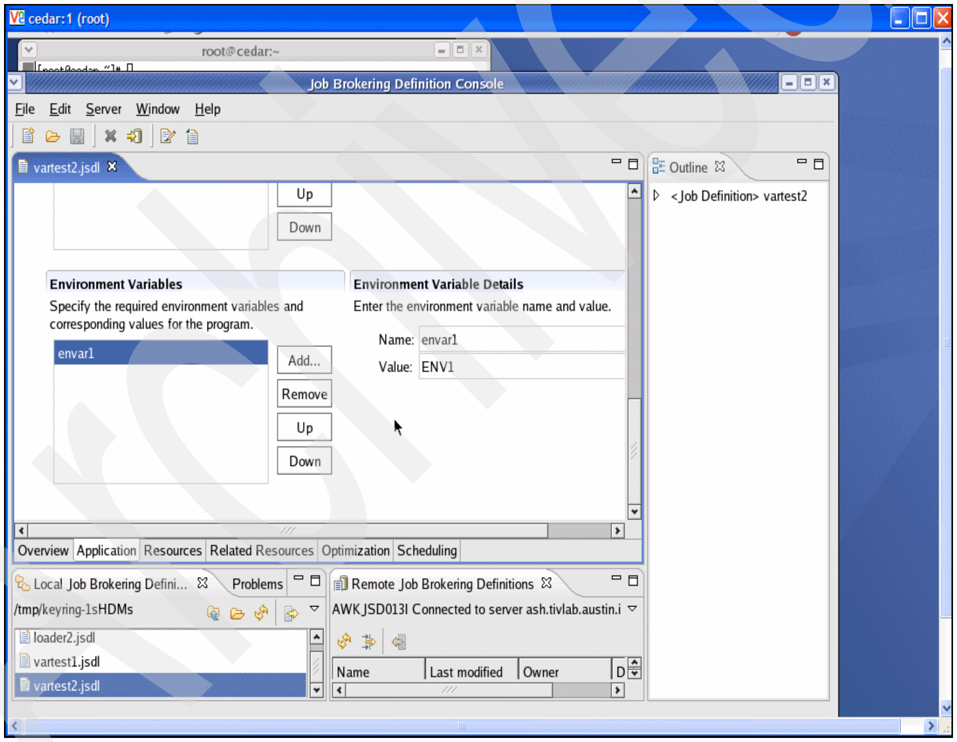
Figure 4-10 Tivoli Dynamic Workload Broker Web Console using Submit to change job variable values for a job submission

4.3.2 Environment variables

Environment variables are set in the run-time environment for the Tivoli Dynamic Workload Broker job definition. Unlike job variables, environment variable values cannot be set in the Tivoli Dynamic Workload Broker Web Console, Tivoli Workload Scheduler Task field, or the Tivoli Dynamic Workload Broker CLI.

Environment variables can be used to change the run-time environment for the job on the assigned resource. This provides flexibility in that you can write a job definition with environment variables so that only those values need to change when you change the resources for the job definition.

Environment variables are referenced as $*varname*, where *varname* is the name of the environment variable. See Figure 4-11.


cedar:1 (root)
root@cedar:~
Job Brokering Definition Console
File Edit Server Window Help
vartest2.jsdl
Outline
<Job Definition> vartest2
Up
Down
Add...
Remove
Up
Down
Overview Application Resources Related Resources Optimization Scheduling
Local Job Brokering Defini...
Problems
Remote Job Brokering Definitions
Name Last modified Owner D

Environment Variables

Specify the required environment variables and corresponding values for the program.

envar1

Environment Variable Details

Enter the environment variable name and value.

Name: envar1

Value: ENV1

/tmp/keyring-1sHDMs

loader2.jsdl
vartest1.jsdl
vartest2.jsdl

AWKJSD013I Connected to server ash.tivlab.austin.i

Figure 4-11 Tivoli Dynamic Workload Broker job definition vartest2 containing environment variable envar1 with value ENV1 as seen from the JBDC

Example 4-6   Job definition vartest2.jsdl file

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
description="testing environment variables" name="vartest2">
  <jsdl:application name="executable">
    <jsdle:executable>
      <jsdle:environment>
        <jsdle:variable name="envar1">ENV1</jsdle:variable>
      </jsdle:environment>
      <jsdle:script>#!/bin/ksh
echo "environmental variable value is $envar1"</jsdle:script>
    </jsdle:executable>
  </jsdl:application>
</jsdl:jobDefinition>
```

Example 4-7   Job output for job vartest2

```
environmental variable value is ENV1
```

4.3.3 Indirect use of job variables in scripts

There is a technique that you can use to reference job variable values in the executable script. You cannot directly reference job variables in the executable scripts because job variables are resolved only in the JSDL statements. Since environment variables are defined in JSDL statements, you can create a job definition in which an environment variable has the value of a job variable and the executable script references the environment variable.

An environment variable can be the link between a job variable value and its use in an executable script. This is an indirect use of job variables in the scripts.

In Example 4-8 the value of job variable *jobvar* is assigned to environmental variable *jevar*. You can send job variable values to the scripts through environment variables.

Example 4-8   Job variable jobvar referenced by a script through environment variable jevar in job definition vartest3

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
```

```
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
description="test of job and environment variables" name="vartest3">
  <jsdl:variables>
    <jsdl:stringVariable
name="jobvar">JobVariable</jsdl:stringVariable>
  </jsdl:variables>
  <jsdl:application name="executable">
    <jsdle:executable>
      <jsdle:environment>
        <jsdle:variable name="jevar">${jobvar}</jsdle:variable>
      </jsdle:environment>
      <jsdle:script>#!/bin/ksh
echo "job variable is $jevar"</jsdle:script>
    </jsdle:executable>
  </jsdl:application>
</jsdl:jobDefinition>
```

Example 4-9 Job output for job vartest3

```
job variable is JobVariable
```

4.4 Planning and choreography

Tivoli Dynamic Workload Broker provides virtualization of resources (physical computers) so that jobs can be assigned a resource that represents a function rather than a particular physical computer. Little will be gained if the computers cannot be grouped together or otherwise treated as functions. If a specific computer will always be used by a specific job then Tivoli Dynamic Workload Broker will not be making any decisions regarding the assignment of that resource to that job.

It is a good idea to have the Tivoli Dynamic Workload Broker server installed on a machine without a Tivoli Dynamic Workload Broker agent. This will minimize contention for resources. For example, if a Tivoli Dynamic Workload Broker server is defined as a Tivoli Dynamic Workload Broker agent then the Tivoli Dynamic Workload Broker server could determine that a job should run on itself. This will increase the utilization of the Tivoli Dynamic Workload Broker server, which could then result in the Tivoli Dynamic Workload Broker server thrashing in an ineffective manner.

The Job Brokering Definition Console and the Web Console could be installed on workstations that are not part of the managed computers. While these components could be installed on the Tivoli Dynamic Workload Broker, they should be installed on a separate machine to minimize contention for resources and clarify what are local job definitions (those developed using the Job Brokering Definition Console and stored on that machine) and what are remote job definitions (those stored on the Tivoli Dynamic Workload Broker server).

Figure 4-12 illustrates an installation involving Tivoli Dynamic Workload Broker and Tivoli Workkload Scheduler.

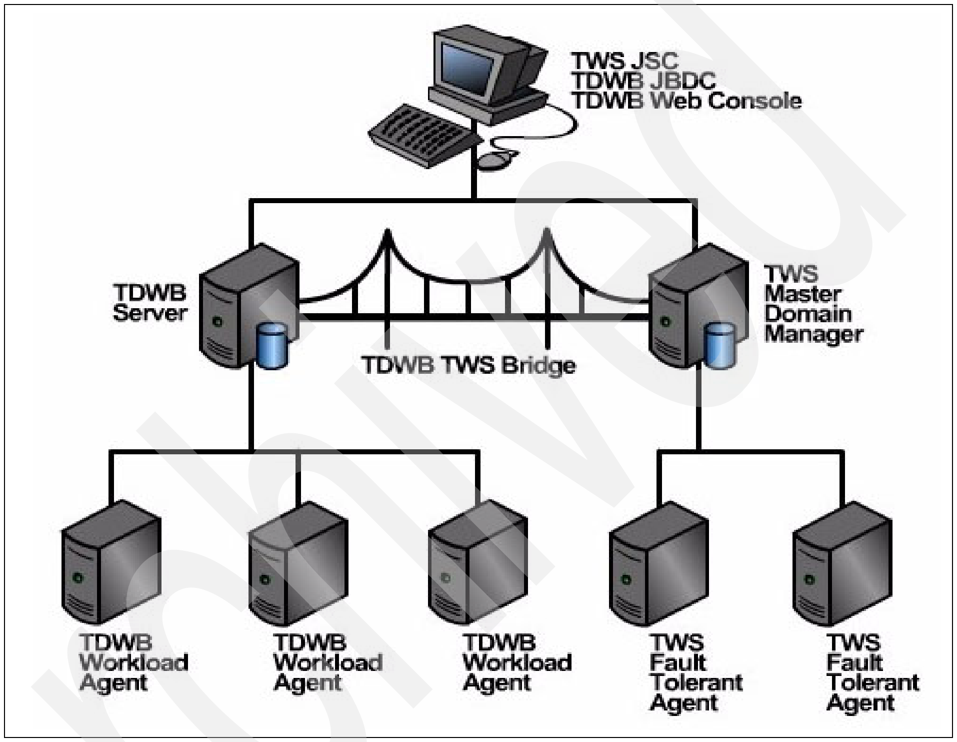


Figure 4-12 Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler connected through the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent

4.4.1 Considerations for Tivoli Workload Scheduler integration

The key points in an integration with Tivoli Workload Scheduler are:

- The Tivoli Dynamic Workload Broker server is configured as a standard agent in the Tivoli Workload Scheduler environment but does not have all of the features of a standard agent. In particular, you can only run a Tivoli Workload Scheduler job definition on this computer if the Task field references a Tivoli Dynamic Workload Broker job definition. The computer is a bridge to the Tivoli Dynamic Workload Broker server and not a computer for running other Tivoli Workload Scheduler job definitions.

- Only jobs meant to run in the Tivoli Dynamic Workload Broker environment are sent to the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent (which is a plug-in on the Tivoli Dynamic Workload Broker server).
- The Tivoli Workload Scheduler job definition in the Task field references the Tivoli Dynamic Workload Broker job definition.

It is best to draw a before and after picture of the workstation architecture and job definitions. Planning and the order of job definition migration is critical to a successful integration.

Existing Tivoli Workload Scheduler jobs

The basic idea for migrating existing Tivoli Workload Scheduler job definitions to Tivoli Dynamic Workload Broker is to create a Tivoli Dynamic Workload Broker job definition for each Tivoli Workload Scheduler job definition that is to run on a computer managed by Tivoli Dynamic Workload Broker. Each existing Tivoli Workload Scheduler job definition is then modified so that it references the new Tivoli Dynamic Workload Broker job definition in the Task field.

You will need to create a new Tivoli Workload Scheduler job definition if the existing one is still in use within the Tivoli Workload Scheduler environment. The affected Tivoli Workload Scheduler job streams need to reference any new Tivoli Workload Scheduler job definitions that are created.

The structure of the Tivoli Workload Scheduler job streams does not change. Only the job definitions change, as described above. The predecessor and successor relationships of your Tivoli Workload Scheduler job streams do not change.

These are the product components and steps that you may follow:

1. Existing Tivoli Workload Scheduler job definitions and job streams are extracted form the Tivoli Workload Scheduler database.
2. The Job Brokering Definition Console is used to create Tivoli Dynamic Workload Broker job definitions from these Tivoli Workload Scheduler job definitions and job streams.
3. The Job Brokering Definition Console is used to create Tivoli Workload Scheduler job definitions that reference the newly created Tivoli Dynamic Workload Broker job definitions.
4. The Job Brokering Definition Console is used to update Tivoli Workload Scheduler job streams to reference the newly created Tivoli Workload Scheduler job definitions.
5. The newly created Tivoli Dynamic Workload Broker job definitions are uploaded to the Tivoli Dynamic Workload Broker server.

6. The newly created Tivoli Workload Scheduler job definitions and updated Tivoli Workload Scheduler job streams are placed into the Tivoli Workload Scheduler database.

Migrating a Tivoli Workload Scheduler fault-tolerant agent

The following is an outline of a strategy in which you may run new Tivoli Dynamic Workload Broker job definitions on a computer that was once a Tivoli Workload Scheduler Fault Tolerant Agent.

To move a computer that is a Tivoli Workload Scheduler Standard Agent from the Tivoli Workload Scheduler environment to the Tivoli Dynamic Workload Broker environment:

1. Remove the Tivoli Workload Scheduler workstation definition from the Tivoli Workload Scheduler master domain manager.
2. Remove the Tivoli Workload Scheduler fault-tolerant agent components from the computer to be managed by Tivoli Dynamic Workload Broker.
3. Install the Tivoli Dynamic Workload Broker Workload agent so that it becomes a Tivoli Dynamic Workload Broker managed computer.
4. Create (or modify) a Tivoli Dynamic Workload Broker logical resource that will reference the computer.
5. Migrate the affected Tivoli Workload Scheduler job streams and job definitions.

Submitting Tivoli Workload Scheduler jobs

Once Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker are integrated, the creation of new job definitions to be run on Tivoli Dynamic Workload Broker managed computers scheduled in Tivoli Workload Scheduler can take place.

Essentially, a Tivoli Workload Scheduler job definition must be created that has a Task field that references a Tivoli Dynamic Workload Broker job definition. The workstation for the Tivoli Workload Scheduler job definition is the machine that has the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler agent installed. This machine looks like a Tivoli Workload Scheduler standard agent to the Tivoli Workload Scheduler master domain manager (or domain manager). When the job definition arrives on this workstation the Tivoli Dynamic Workload Broker server uses the value in the Task field as the name of a Tivoli Dynamic Workload Broker job definition. It then retrieves the Tivoli Dynamic Workload Broker job definition from the Tivoli Dynamic Workload Broker job repository, assigns a resource, and runs the job. Job status is reported back to Tivoli Workload Scheduler through the Tivoli Dynamic Workload Broker.

The following steps describe submitting a Tivoli Workload Scheduler job:

1. Create a job definition in Tivoli Dynamic Workload Broker (TDWBjob).
2. Create a job definition in Tivoli Workload Scheduler (TWSjob - the *target workstation* of the Tivoli Workload Scheduler job definition is where the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent lives). The Task String section of the Tivoli Workload Scheduler job definition specifies the name of the Tivoli Dynamic Workload Broker job definition (TDWBjob).
3. TWSjob is scheduled to run in Tivoli Workload Scheduler (from a job stream or as an *ad hoc* job submission).
4. TWSjob arrives at the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent workstation to run. (This looks like a Tivoli Workload Scheduler Standard Agent to Tivoli Workload Scheduler.)
5. Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent sees the Tivoli Dynamic Workload Broker job definition TDWBjob specified in the Task string.
6. Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent tells Tivoli Dynamic Workload Broker server to run the TDWBjob.
7. Tivoli Dynamic Workload Broker server retrieves the TDWBjob definition from the job repository, assigns a resource, and runs the job.
8. The Tivoli Dynamic Workload Broker server communicates status to the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent.
9. The job status can be monitored from the Tivoli Workload Scheduler.

Converting Tivoli Workload Scheduler jobs

After integration of Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker you may find a need to convert an existing Tivoli Workload Scheduler job definition that runs on a Tivoli Workload Scheduler agent to a Tivoli Workload Scheduler job definition that references a Tivoli Dynamic Workload Broker job definition to run on a Tivoli Dynamic Workload Broker managed computer.

The following steps are the details for this conversion:

1. Export Tivoli Workload Scheduler job definitions and job streams to files using the `composer` command.
2. Save the files on a workstation that has the Job Brokering Definition Console installed.
3. Use the Job Brokering Dedfinition Console function **Import From Tivoli Workload Scheduler And Tivoli Dynamic Workload Broker** and specify the Tivoli Workload Scheduler job definition and job stream files for the import.

4. Contents of the Tivoli Workload Scheduler jobs are displayed in the Tivoli Workload Scheduler Jobs table.
5. Associate the jobs to a logical resource using the Tivoli Dynamic Workload Broker Logical Resources pane. (This tool adds the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent name as a prefix to the job name.)
6. Use **Export to Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker** to create the new Tivoli Dynamic Workload Broker job definition.
   a. Specify the name of the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent (bridge).
   b. Specify the file name for the exported job definitions.
   c. Specify the file name for the exported job streams.
7. Edit the Job Submission Definition Language files that were saved on the local workstation to add Tivoli Dynamic Workload Broker specific features (if needed).
8. Save the Tivoli Workload Scheduler job definition and job stream files to the Tivoli Workload Scheduler Server using the `composer` command.
9. Save the new Tivoli Dynamic Workload Broker job definition (files with the *jsdl* extension) in the Tivoli Dynamic Workload Broker repository by uploading to the Tivoli Dynamic Workload Broker server.
10. Schedule the updated job streams from the Tivoli Workload Scheduler.
11. Jobs will be submitted to the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent where the Tivoli Dynamic Workload Broker server is also running.
12. The Tivoli Dynamic Workload Broker server determines where the jobs run by the resources required for each job (specified by the logical resource used in creating the Tivoli Dynamic Workload Broker job definition).

4.5 Resource matching criteria

Tivoli Dynamic Workload Broker assigns resources to job definitions. A resource is a computer (also known as a workstation) where the executable referenced in the job definition runs. The job definition may also optionally specify criteria for optimization that will affect the resource assigned to the job definition. This criteria is used to match the job definition with the resource by Tivoli Dynamic Workload Broker.

The Job Brokering Definition Console is a good place to start creating a job definition that will specify resource matching criteria. The job definition file can be edited later using the Tivoli Dynamic Workload Broker Web Console. Outside of the Job Brokering Definition Console one needs to enter XML code directly into the job definition.

4.5.1 Optimization objective type

The Optimization tab in the Job Brokering Definition Console can be used to establish objectives for the optimization policy. Selection objective will create an *objective element* in the job definition. The Tivoli Dynamic Workload Broker runs the job on the resource matching the optimization requirement. This element is mutually exclusive with the *ewlm element*.

The resource Type can be one of the following with an associated resource property and optimization objective:

- Computer System
  - CPU Utilization
    - Maximize, Minimize
  - Number of Processors
    - Maximize, Maximize Utilization, Minimize, Minimize Utilization
  - Processing Speed
    - Maximize, Minimize
- File System
  - Available
    - Maximize, Minimize
  - Total Storage Capacity
    - Maximize, Maximize Utilization, Minimize, Minimize Utilization
- Logical Resource
  - Quantity
    - Maximize, Maximize Utilization, Minimize, Minimize Utilization
- Operating System
  - Free Physical Memory
    - Maximize, Minimize
  - Free Swap Space
    - Maximize, Minimize
  - Free Virtual Memory
    - Maximize, Minimize
  - Total Physical Memory
    - Maximize, Maximize Utilization, Minimize, Minimize Utilization
  - Total Swap Space
    - Maximize, Maximize Utilization, Minimize, Minimize Utilization
  - Total Virtual Memory
    - Maximize, Maximize Utilization, Minimize, Minimize Utilization

4.5.2 Optimization Enterprise Workload Manager type

Selection of this type creates an *ewlm element* in the job definition. This element specifies the optimization based on Enterprise Workload Manager resource weight calculation. The Tivoli Dynamic Workload Broker runs the job on the best available resources as indicated by Enterprise Workload Manager. This element is mutually exclusive with the *objective element*.

4.5.3 Resources

Resources can be specified in the Job Brokering Definition Console (using the Resources tab) as:

- Hardware requirements
- Software requirements
- Advanced requirements

Hardware requirements

You can specify the following for the hardware requirements:

- Physical memory (exact or range of values)
- Virtual memory (exact or range of values)
- Candidate hosts
- Candidate CPU architecture (exact or range of speed values)

Software requirements

You can specify the following for the software requirements:

- Candidate operating system
- Logical resource quantity
- File systems (exact or range of disk space)

The logical resource quantity is an arbitrary value that you set so that the logical resource is a consumable resource. The quantity is returned when the job completes.

Advanced requirements

You can specify *resource properties* with AND and OR logic. The resource properties are:

- CPU utilization
- Host name
- Manufacturer
- Model
- Number of processors
- Processing speed
- Processor type

Each of these resource properties is specified with an exact value or a range value.

A relationship of *Associates With* or *Contains* can be specified for the resource where the job is to be run. An allocation for the number of processors for the resource can be specified. Resource groups to which the resource must belong can be specified.

4.5.4 Related resources

A *related resource* is a resource identified by an *ID* that is referenced in a resource element. Logical resources are typical *related resources*. See Example 4-4 on page 146 for a job definition that uses a logical resource group as a related resource.

The related resource type can be:

- Computer system
- File system
- Logical resource
- Network system
- Operating system

The resource properties can be one of the following with an exact or a range of values:

- Display name
- Quantity
- Sub type

You can specify a relationship for *Associates With* or *Contains*.

An allocation quantity is an arbitrary value that you set so that the related resource is a consumable resource. The quantity is returned when the job completes.

Resource groups to which the resource must belong can be specified.

4.6 Monitoring computers and jobs

Computers and jobs can be monitored from the Tivoli Dynamic Workload Broker Web Console. The Tivoli Dynamic Workload Broker Web Console shows the status of computers managed by Tivoli Dynamic Workload Broker. The network service *Agent Manager* provides the Tivoli Dynamic Workload Broker server with the information about the status of the computers managed by the Tivoli Dynamic Workload Broker server.

Jobs can be monitored from the:

- Tivoli Workload Scheduler JSC (Job Submission Console) and CLI
- Tivoli Dynamic Workload Broker Web Console
- Tivoli Dynamic Workload Broker CLI

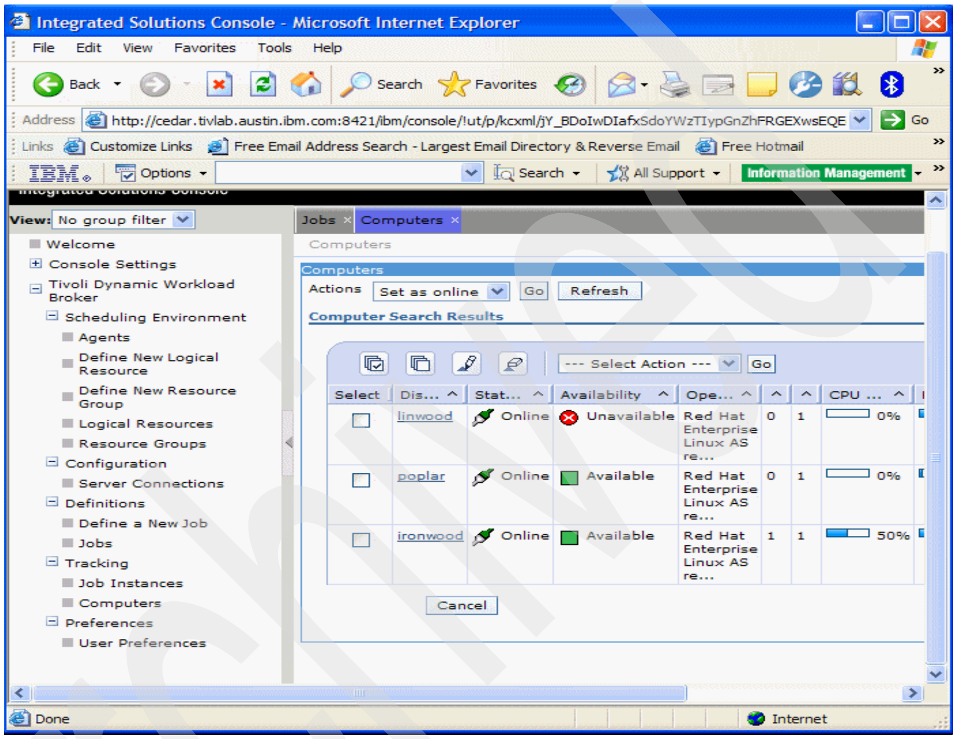
Note: The status for Tivoli Dynamic Workload Broker jobs as seen the in Tivoli Dynamic Workload Broker Web Console will be similar, but not identical, to the status for the same jobs as seen using the Tivoli Dynamic Workload Broker CLI.

Tivoli Workload Scheduler Job Submission Console and CLI

The Tivoli Workload Scheduler Job Submission Console (JSC) provides information about the status of the Tivoli Workload Scheduler job running on the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent. The Tivoli Dynamic Workload Broker job itself runs on a Tivoli Dynamic Workload Broker managed computer. You can browse the job output in Tivoli Workload Scheduler to see the name of the Tivoli Dynamic Workload Broker managed computer used for the Tivoli Dynamic Workload Broker job that was referenced in the Tivoli Workload Scheduler job definition. Tivoli Workload Scheduler also has a CLI to monitor the Tivoli Workload Scheduler job and browse the job output.

Tivoli Dynamic Workload Broker Web Console

The Tivoli Dynamic Workload Broker Web Console shows the status of Tivoli Dynamic Workload Broker jobs and Tivoli Dynamic Workload Broker managed computers. It does not show the status of Tivoli Workload Scheduler jobs and Tivoli Workload Scheduler workstations. See Figure 4-13.


Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Links Customize Links Free Email Address Search - Largest Email Directory & Reverse Email Free Hotmail
IBM Options Search All Support Information Management
View: No group filter
Welcome
Console Settings
Tivoli Dynamic Workload Broker
Scheduling Environment
Agents
Define New Logical Resource
Define New Resource Group
Logical Resources
Resource Groups
Configuration
Server Connections
Definitions
Define a New Job
Jobs
Tracking
Job Instances
Computers
Preferences
User Preferences
Jobs Computers
Actions Set as online Go Refresh
--- Select Action --- Go
Cancel
Done Internet

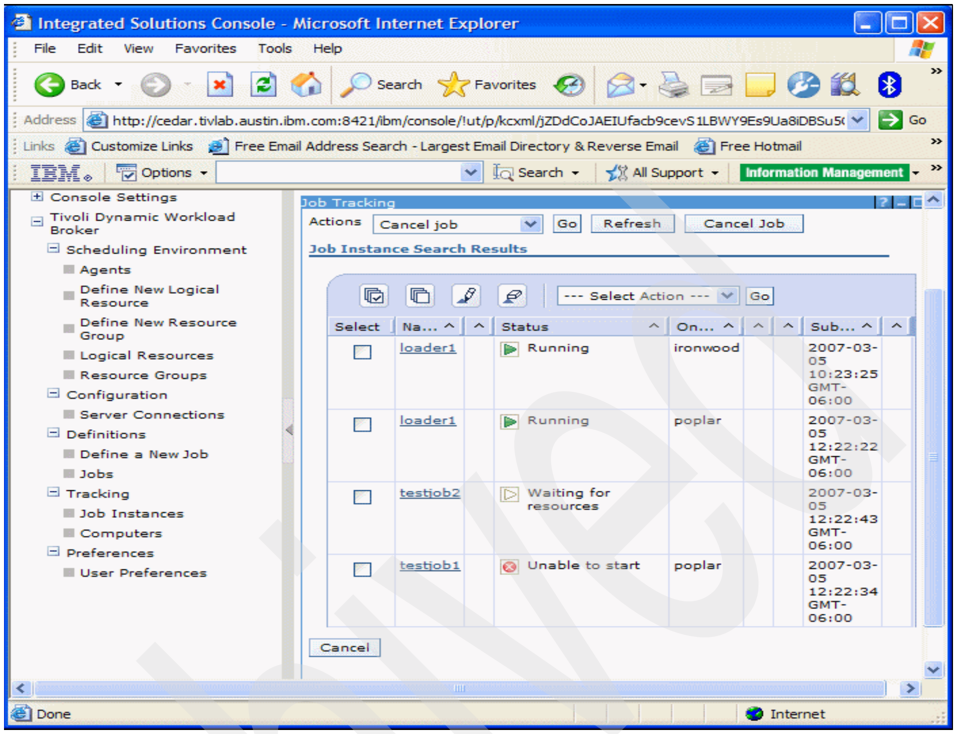
Address http://cedar.tivlab.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/jY_BDoIwDIafxSdoYWzTIypGnZhFRGEXwsEQE

Computers

Computer Search Results

Select	Dis...	Stat...	Availability	Ope...			CPU ...
☐	linwood	Online	Unavailable	Red Hat Enterprise Linux AS re...	0	1	0%
☐	poplar	Online	Available	Red Hat Enterprise Linux AS re...	0	1	0%
☐	ironwood	Online	Available	Red Hat Enterprise Linux AS re...	1	1	50%

Figure 4-13 Tivoli Dynamic Workload Broker Web Console showing status of the Tivoli Dynamic Workload Broker managed computers

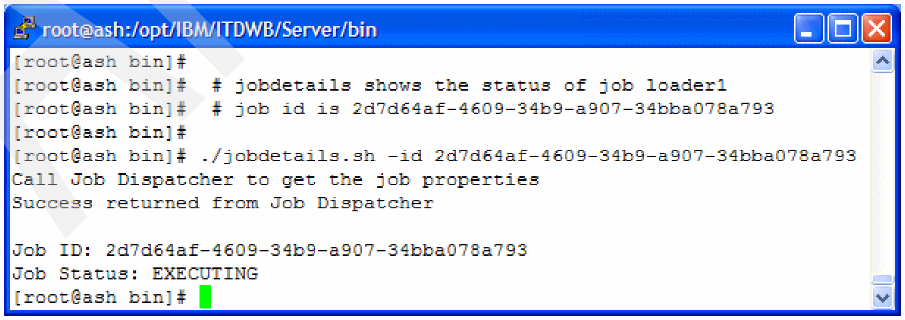

Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://cedar.tivlab.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/jZDdCoJAEIUfacb9cevS1LBWY9Es9Ua8iDBSu5(Go
Links Customize Links Free Email Address Search - Largest Email Directory & Reverse Email Free Hotmail
IBM Options Search All Support Information Management
Console Settings
Tivoli Dynamic Workload Broker
Scheduling Environment
Agents
Define New Logical Resource
Define New Resource Group
Logical Resources
Resource Groups
Configuration
Server Connections
Definitions
Define a New Job
Jobs
Tracking
Job Instances
Computers
Preferences
User Preferences
Job Tracking
Actions Cancel job Go Refresh Cancel Job
Job Instance Search Results
--- Select Action --- Go
Cancel
Done
Internet

Select	Na...	Status	On...	Sub...
☐	loader1	Running	ironwood	2007-03-05 10:23:25 GMT-06:00
☐	loader1	Running	poplar	2007-03-05 12:22:22 GMT-06:00
☐	testjob2	Waiting for resources		2007-03-05 12:22:43 GMT-06:00
☐	testjob1	Unable to start	poplar	2007-03-05 12:22:34 GMT-06:00

Figure 4-14 Tivoli Dynamic Workload Broker Web Console showing status of the Tivoli Dynamic Workload Broker jobs

Tivoli Dynamic Workload Broker CLI

The Tivoli Dynamic Workload Broker CLI jobdetails, jobgetexecutionlog, and jobquery scripts can be used to monitor Tivoli Dynamic Workload Broker jobs, as shown in Figure 4-15. You must run the Tivoli Dynamic Workload Broker CLI scripts on the Tivoli Dynamic Workload Broker server machine.



```
[root@ash bin]#
[root@ash bin]#  # jobdetails shows the status of job loader1
[root@ash bin]#  # job id is 2d7d64af-4609-34b9-a907-34bba078a793
[root@ash bin]#
[root@ash bin]# ./jobdetails.sh -id 2d7d64af-4609-34b9-a907-34bba078a793
Call Job Dispatcher to get the job properties
Success returned from Job Dispatcher

Job ID: 2d7d64af-4609-34b9-a907-34bba078a793
Job Status: EXECUTING
[root@ash bin]#
```

Figure 4-15 Tivoli Dynamic Workload Broker CLI jobdetails -id showing status of a job

5

Advanced guide to Tivoli Dynamic Workload Broker

Tivoli Dynamic Workload Broker is a key element in a comprehensive on demand Tivoli Workload Automation family portfolio. In this chapter we discuss the following topics:

- "Tivoli Workload Scheduler migration to Tivoli Dynamic Workload Broker" on page 176
- "Job Submission Description Language reference" on page 192
- "Tivoli Dynamic Workload Broker user authorization and authentication" on page 218
- "Command-line interface" on page 232

5.1 Tivoli Workload Scheduler migration to Tivoli Dynamic Workload Broker

This section describes in detail how you can migrate a Tivoli Workload Scheduler job that runs on a Tivoli Workload Scheduler workstation to a Tivoli Dynamic Workload Broker job that runs on a computer managed by Tivoli Dynamic Workload Broker.

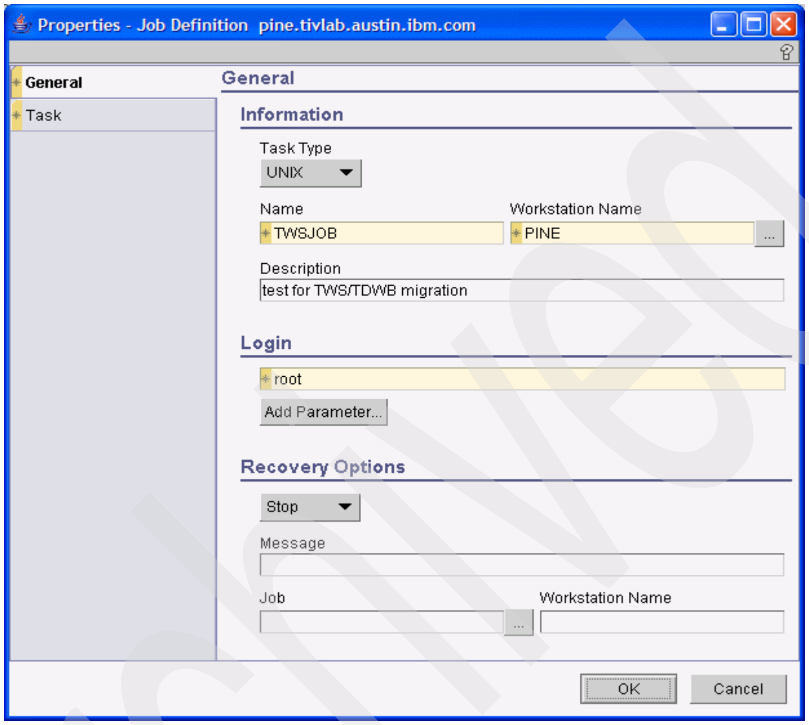
After migration the Tivoli Workload Scheduler will be used to run a new Tivoli Workload Scheduler job. The end result will be a Tivoli Dynamic Workload Broker job running on a computer managed by Tivoli Dynamic Workload Broker.

The basic idea is that the command or script run by the Tivoli Workload Scheduler job stream will run from a new Tivoli Dynamic Workload Broker job definition. A new Tivoli Workload Scheduler job definition will reference the new Tivoli Dynamic Workload Broker job definition. Tivoli Workload Scheduler runs the Tivoli Workload Scheduler on the Tivoli Dynamic Workload Broker server, which then assigns a resource to the Tivoli Dynamic Workload Broker job definition.

The migration results in the addition of an intermediate step in which *Tivoli Workload Scheduler* communicates with *Tivoli Dynamic Workload Broker* through the *Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent*.

5.1.1 Initial Tivoli Workload Scheduler job definition and job stream

We begin with a Tivoli Workload Scheduler job definition named *TWSJOB* that runs the command **`dir`**, as shown in Figure 5-1 and Figure 5-2 on page 178.


Properties - Job Definition pine.tivlab.austin.ibm.com

General
Task

General

Information

Task Type: UNIX

Name: TWSJOB

Workstation Name: PINE

Description: test for TWS/TDWB migration

Login

root

Recovery Options

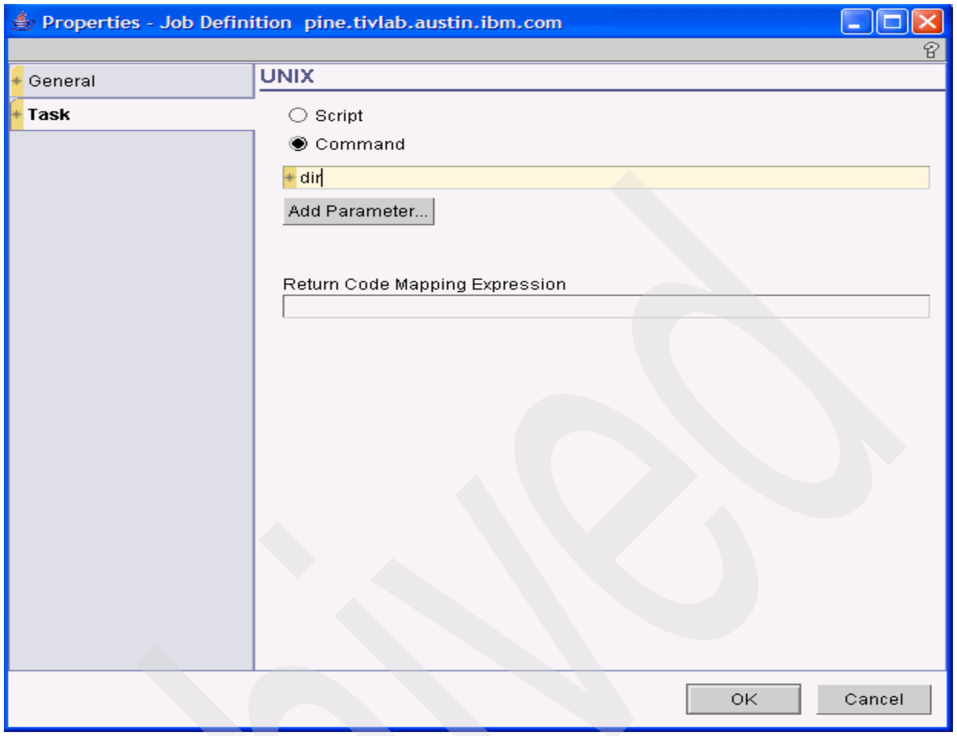
Stop

Message

Job

Workstation Name

Figure 5-1 TWSJOB job definition - General tab



Command: dir

Return Code Mapping Expression

Figure 5-2 TWSJOB job definition - Task tab

In Tivoli Workload Scheduler one can submit a job definition (which is automatically put in a job stream), submit a job stream, or schedule a job stream to run. For this discussion, the job stream containing job definition TWSJOB is TWSJOBSTREAM. TWSJOB is the only job definition in job stream TWSJOBSTREAM.

Tivoli Dynamic Workload Broker has a migration utility to migrate TWS job definitions and their related job streams into Tivoli Workload Scheduler job definitions and job streams that result in Tivoli Dynamic Workload Broker job definitions running on a computer managed by Tivoli Dynamic Workload Broker.

Note: The Tivoli Dynamic Workload Broker migration utility for Tivoli Workload Scheduler job definitions requires a related Tivoli Workload Scheduler job stream. For this reason you may need to create a Tivoli Workload Scheduler job stream to use in the migration of each Tivoli Workload Scheduler job definition that does not have a job stream.

Some Tivoli Workload Scheduler job streams contain more than one job definition. You may use the same job stream for each of these contained job definitions in the Tivoli Dynamic Workload Broker migration tool. Job definitions that will not be migrated will be left alone by the migration tool.

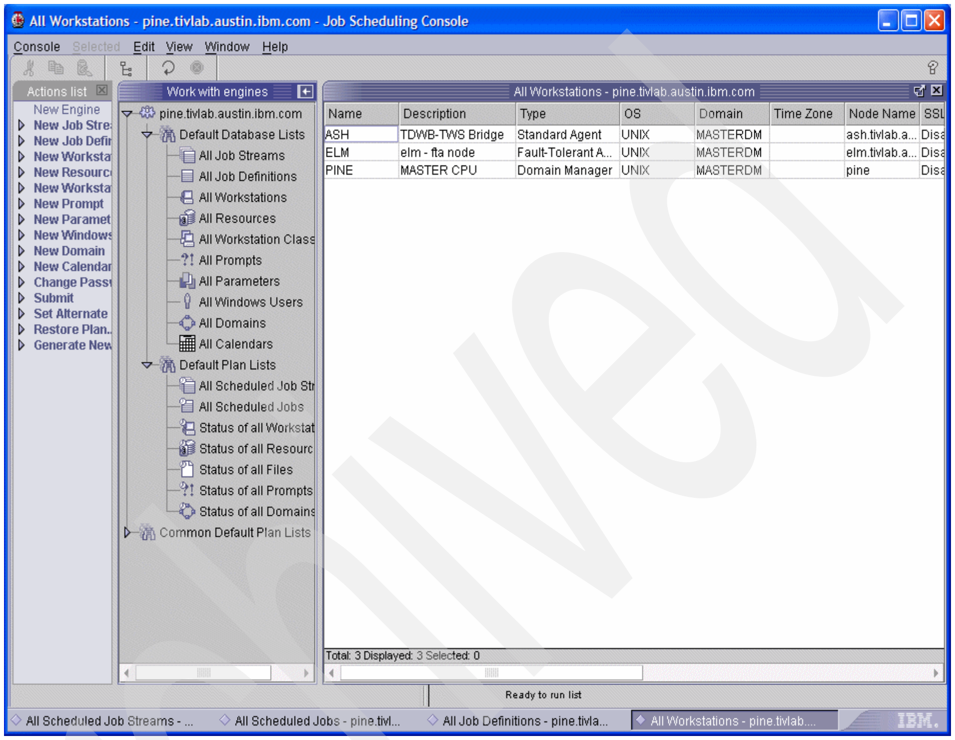
For this discussion the job stream TWSJOBSTREAM containing job definition TWSJOB runs on the Tivoli Workload Scheduler fault tolerant agent (FTA) known as PINE. The name of the workstation is used by the migration utility to name the new Tivoli Dynamic Workload Scheduler job definition.

5.1.2 Situation after migration to Tivoli Dynamic Workload Broker

The desired situation after migration to Tivoli Dynamic Workload Broker is to have a Tivoli Dynamic Workload Broker job definition (called PINE_TWSJOB for this discussion) that runs the command that TWSJOB was running (**`dir`**) on a computer controlled by Tivoli Dynamic Workload Broker.

In this example scenario the Tivoli Dynamic Workload Broker server and the Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent run on the ASH machine. This machine is initially not part of the Tivoli Workload Scheduler environment.

The ASH machine must be configured as a standard agent for the Tivoli Workload Scheduler implementation. This is not a true standard agent in that only Tivoli Workload Scheduler job definitions that reference in the Task field a Tivoli Dynamic Workload Broker job definition run on ASH. See Figure 5-3.



Name	Description	Type	OS	Domain	Time Zone	Node Name	SSL
ASH	TDWB-TWS Bridge	Standard Agent	UNIX	MASTERDM		ash.tivlab.a...	Disa
ELM	elm - fta node	Fault-Tolerant A...	UNIX	MASTERDM		elm.tivlab.a...	Disa
PINE	MASTER CPU	Domain Manager	UNIX	MASTERDM		pine	Disa

Total: 3 Displayed: 3 Selected: 0

Figure 5-3 All workstations with ASH as the bridge between Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker

After migration you will have this situation:

- ASH (the computer with the Tivoli Dynamic Workload Broker server) will be configured as a Standard Agent in the Tivoli Workload Scheduler environment.
- The Task field of the new Tivoli Workload Scheduler job definition PINE_TWSJOB (created by the migration function) references a new Tivoli Dynamic Workload Scheduler job definition PINE_TWSJOB (also created by the migration function).
- The newly created Tivoli Dynamic Workload Broker job definition PINE_TWSJOB runs the command **`dir`**.

- An updated Tivoli Workload Scheduler job definition TWSJOBSTREAM that runs the newly created Tivoli Workload Scheduler job definition PINE_TWSJOB.
- A logical resource (TWSLOGICALRESOURCE for this discussion, created by you) referencing the Tivoli Dynamic Workload Broker managed computers for PINE_TWSJOB.

Note: The name of the newly created Tivoli Workload Scheduler job definition and the Tivoli Dynamic Workload Scheduler definition will be the same. The name will be <*workstation*>_<*job definition*>, where *workstation* is the Tivoli Workload Scheduler workstation where the Tivoli Workload Scheduler job definition was running.

When you submit TWSJOBSTREAM after migration:

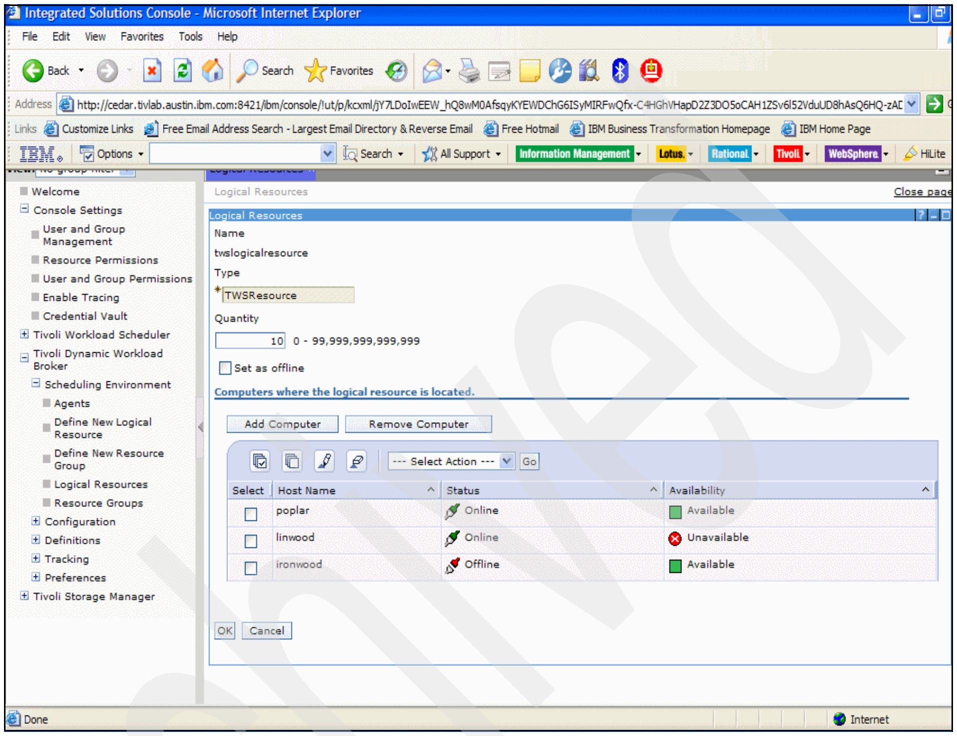
- Tivoli Workload Scheduler will cause PINE_TWSJOB to run on the ASH machine.
- Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent on the ASH machine will find the Tivoli Dynamic Workload Broker job definition PINE_TWSJOB in the Task field of PINE_TWSJOB and pass that on to the Tivoli Dynamic Workload Broker server.
- Tivoli Dynamic Workload Broker server will assign PINE_TWSJOB to a computer in the logical resource TWSLOGICALRESOURCE.
- The command `dir` will run on one of the machines in TWSLOGICALRESOURCE assigned by Tivoli Dynamic Workload Broker server.

5.1.3 Create logical resources for the new job definitions

All Tivoli Workload Scheduler job definitions that reference (in the Task field) Tivoli Dynamic Workload Broker job definitions will run on the Tivoli Dynamic Workload Broker computer (ASH in this discussion). The Tivoli Dynamic Workload Broker Tivoli Workload Scheduler Agent is the bridge that sends the Tivoli Dynamic Workload Broker job definition name to the Tivoli Dynamic Workload Broker server for assignment to a resource.

The Tivoli Dynamic Workload Broker job definition has an associated logical resource that contains the desired computers for running the job.

In this discussion the logical resource TWSLOGICALRESOURCE is created by you and contains the desired computers for running PINE_TWSJOB, as shown in Figure 5-4.



http://cedar.tivlab.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/jY7LDoIwEEW_hQ8wM0AfsqyKYEWDChG6ISyMIRFwQfx-C4HGhVHapD2Z3DO5oCAH1ZSv6l52VduUD8hAsQ6HQ-zAE

Logical Resources

Name
twslogicalresource

Type
*TWSResource

Quantity
10 0 - 99,999,999,999,999

Set as offline

Computers where the logical resource is located.

Select	Host Name	Status	Availability
☐	poplar	Online	🟩 Available
☐	linwood	Online	✖ Unavailable
☐	ironwood	Offline	🟩 Available

Figure 5-4 TWSLOGICALRESOURCE - a logical resource containing the three computers poplar, linwood, and ironwood

In practice you may want to use several different computers for different Tivoli Dynamic Workload Broker job definitions. This is done by creating one logical resource for each collection of computers.

Note: You need a logical resource for each unique set of computers required by the Tivoli Dynamic Workload Broker job definitions arriving from Tivoli Workload Scheduler.

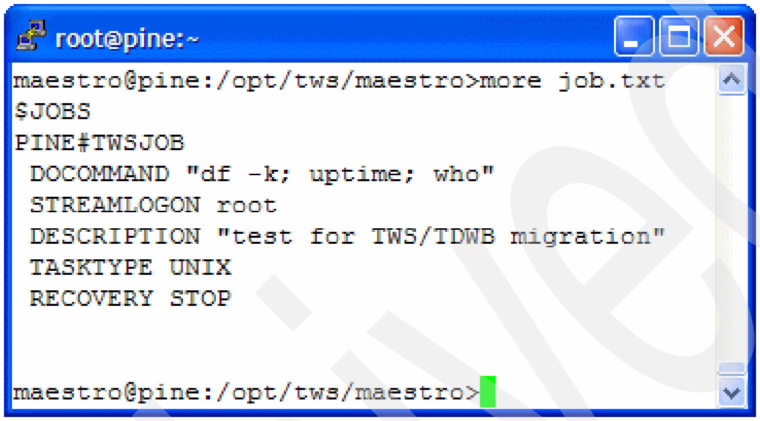
5.1.4 Extract Tivoli Workload Scheduler job definitions and job streams

The job definitions and the associated job streams are extracted to files using the Tivoli Workload Scheduler **composer** command. In this discussion the Tivoli

Workload Scheduler job definitions are extracted to the file *job.txt* and the Tivoli Workload Scheduler job streams are extracted to the file *sched.txt*:

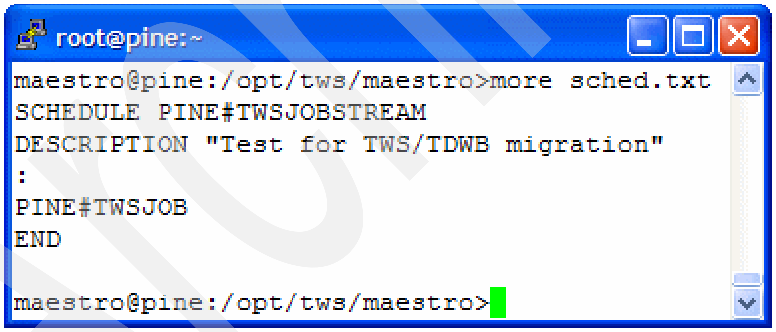
```
create job.txt from job=TWSJOB
create sched.txt from jobstream=TWSJOBSTREAM
```

The files *job.txt* and *sched.txt* are then moved to a machine that is running the Job Brokering Definition Console (JBDC). See Figure 5-5 and Figure 5-6.



```
maestro@pine:/opt/tws/maestro>more job.txt
$JOBS
PINE#TWSJOB
 DOCOMMAND "df -k; uptime; who"
 STREAMLOGON root
 DESCRIPTION "test for TWS/TDWB migration"
 TASKTYPE UNIX
 RECOVERY STOP


maestro@pine:/opt/tws/maestro>
```

Figure 5-5 job.txt file contents for TWSJOB job definition



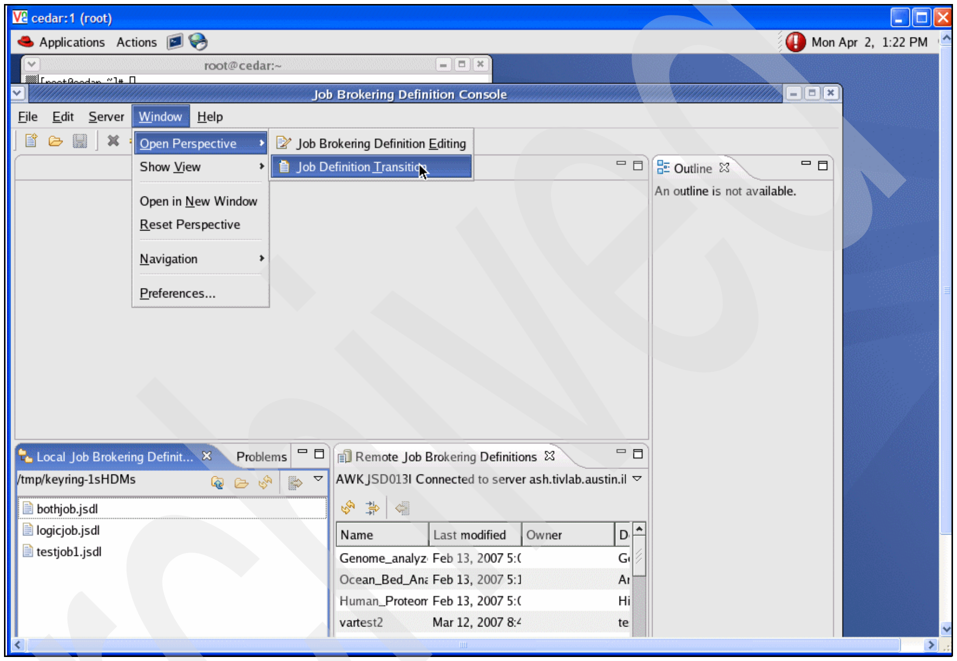
```
maestro@pine:/opt/tws/maestro>more sched.txt
SCHEDULE PINE#TWSJOBSTREAM
DESCRIPTION "Test for TWS/TDWB migration"
:
PINE#TWSJOB
END

maestro@pine:/opt/tws/maestro>
```

Figure 5-6 sched.txt file contents for TWSJOBSTREAM job stream

5.1.5 Import Tivoli Workload Scheduler job definitions and job streams to Tivoli Dynamic Workload Broker

Do the following to import Tivoli Workload Scheduler job definitions and job streams to Tivoli Dynamic Workload Broker:

1. On the Job Brokering Definition Console select **Window** → **Open Perspective** → **Job Definition Transition**, as seen in Figure 5-7.


cedar:1 (root)
Applications Actions
Mon Apr 2, 1:22 PM
root@cedar:~
Job Brokering Definition Console
File Edit Server Window Help
Open Perspective
Show View
Open in New Window
Reset Perspective
Navigation
Preferences...
Job Brokering Definition Editing
Job Definition Transition
Outline
Local Job Brokering Definit...
Problems
Remote Job Brokering Definitions

An outline is not available.

/tmp/keyring-1sHDMs

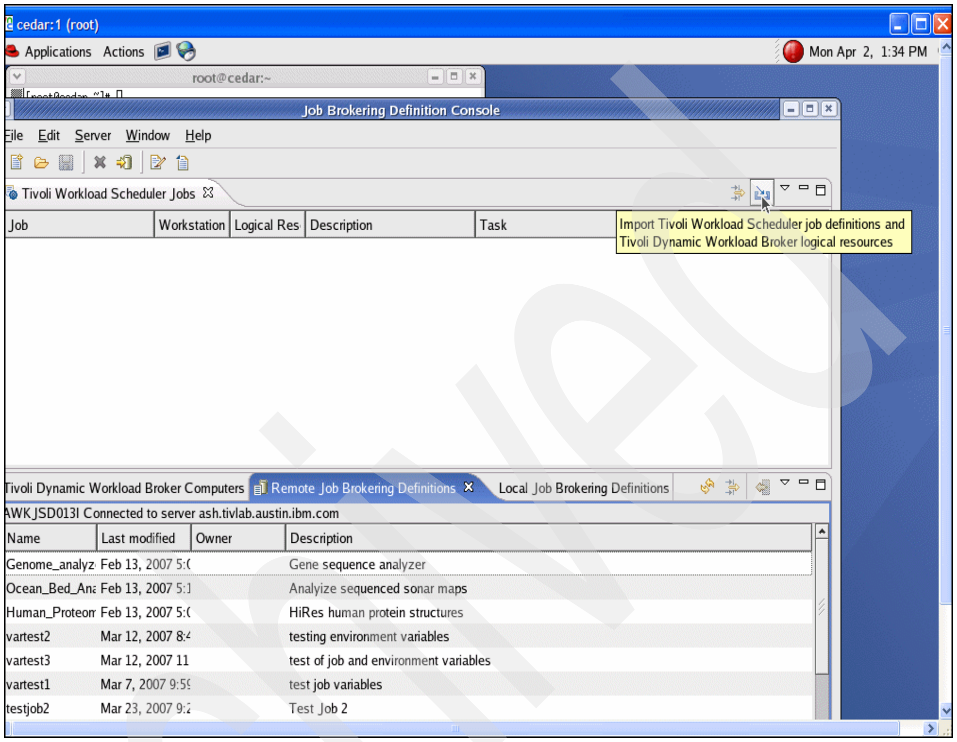
bothjob.jsdl
logicjob.jsdl
testjob1.jsdl

AWKJSD013I Connected to server ash.tivlab.austin.il

Name	Last modified	Owner	D
Genome_analyz	Feb 13, 2007 5:(		G
Ocean_Bed_Ana	Feb 13, 2007 5:1		Ar
Human_Proteon	Feb 13, 2007 5:(		Hi
vartest2	Mar 12, 2007 8:4		te

Figure 5-7 Job Definition Transition takes you to the migration utility

2. On the Tivoli Workload Scheduler Jobs, click the icon for **Import From Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker**. See Figure 5-8.


cedar:1 (root)
Applications Actions
Mon Apr 2, 1:34 PM
root@cedar:~
Job Brokering Definition Console
File Edit Server Window Help
Tivoli Workload Scheduler Jobs
Job Workstation Logical Res Description Task
Tivoli Dynamic Workload Broker Computers
Remote Job Brokering Definitions
Local Job Brokering Definitions

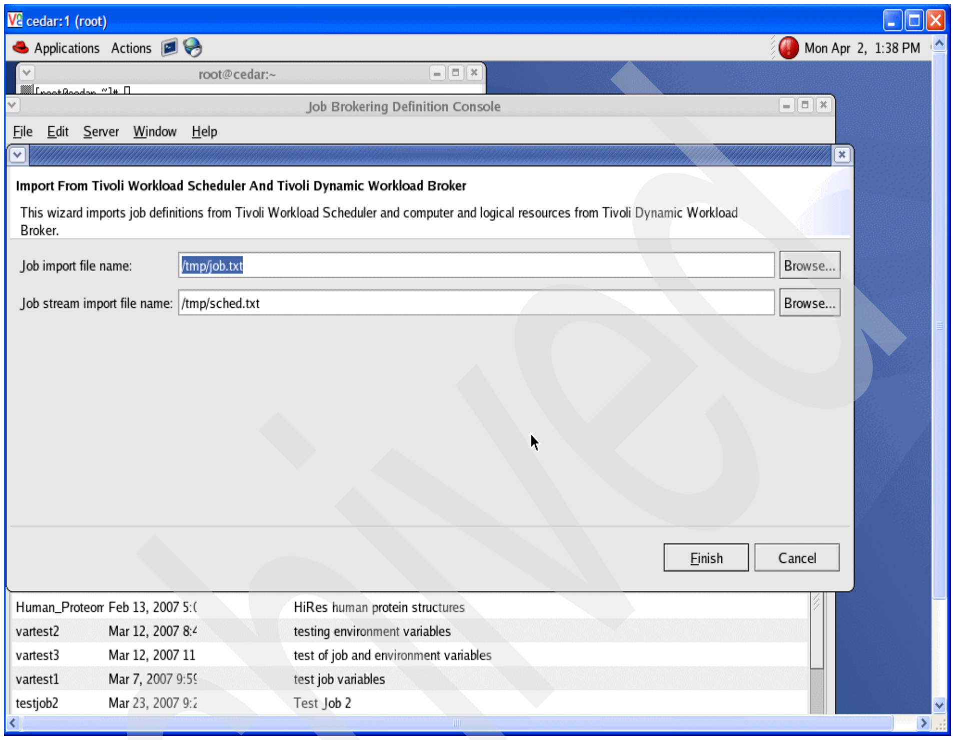
Import Tivoli Workload Scheduler job definitions and Tivoli Dynamic Workload Broker logical resources

AWKJSD013I Connected to server ash.tivlab.austin.ibm.com

Name	Last modified	Owner	Description
Genome_analyz	Feb 13, 2007 5:(		Gene sequence analyzer
Ocean_Bed_Ana	Feb 13, 2007 5:1		Analyze sequenced sonar maps
Human_Proteor	Feb 13, 2007 5:(		HiRes human protein structures
vartest2	Mar 12, 2007 8:4		testing environment variables
vartest3	Mar 12, 2007 11		test of job and environment variables
vartest1	Mar 7, 2007 9:5		test job variables
testjob2	Mar 23, 2007 9:2		Test Job 2

Figure 5-8 Icon for importing Tivoli Workload Scheduler job definitions and job streams

3. Specify the files containing the Tivoli Workload Scheduler job definitions (job.txt for this discussion) and the Tivoli Workload Broker job stream definitions (sched.txt for this discussion), as shown in Figure 5-9.



Import From Tivoli Workload Scheduler And Tivoli Dynamic Workload Broker

This wizard imports job definitions from Tivoli Workload Scheduler and computer and logical resources from Tivoli Dynamic Workload Broker.

Job import file name: /tmp/job.txt

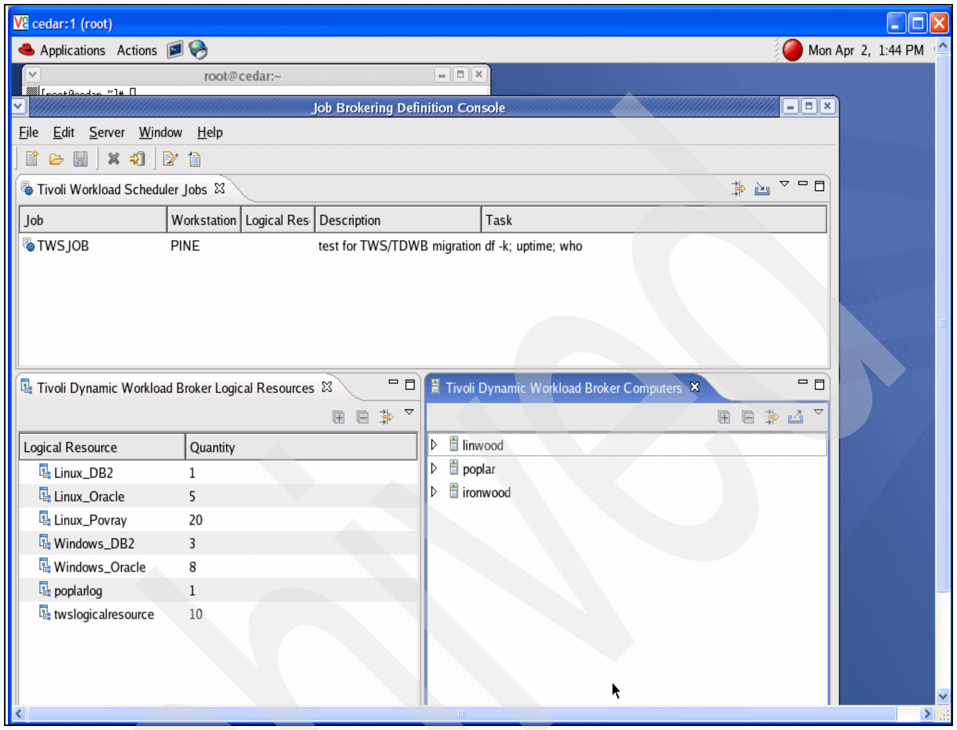
Job stream import file name: /tmp/sched.txt

Human_Proteon	Feb 13, 2007 5:(	HiRes human protein structures
vartest2	Mar 12, 2007 8:4	testing environment variables
vartest3	Mar 12, 2007 11	test of job and environment variables
vartest1	Mar 7, 2007 9:5(	test job variables
testjob2	Mar 23, 2007 9:2	Test Job 2

Figure 5-9 Specify job.txt and sched.txt files for the Tivoli Workload Scheduler job definition and job stream

4. Click **Finish**. The Tivoli Workload Scheduler Jobs window should appear and TWSJOB should be listed under Jobs.

5. You may need to select **Window** → **Reset Perspective** in order to see the Tivoli Dynamic Workload Broker Logical Resources. See Figure 5-10.


cedar:1 (root)
Applications Actions
Mon Apr 2, 1:44 PM
root@cedar:~
Job Brokering Definition Console
File Edit Server Window Help
Tivoli Workload Scheduler Jobs
Tivoli Dynamic Workload Broker Logical Resources
Tivoli Dynamic Workload Broker Computers
linwood
poplar
ironwood

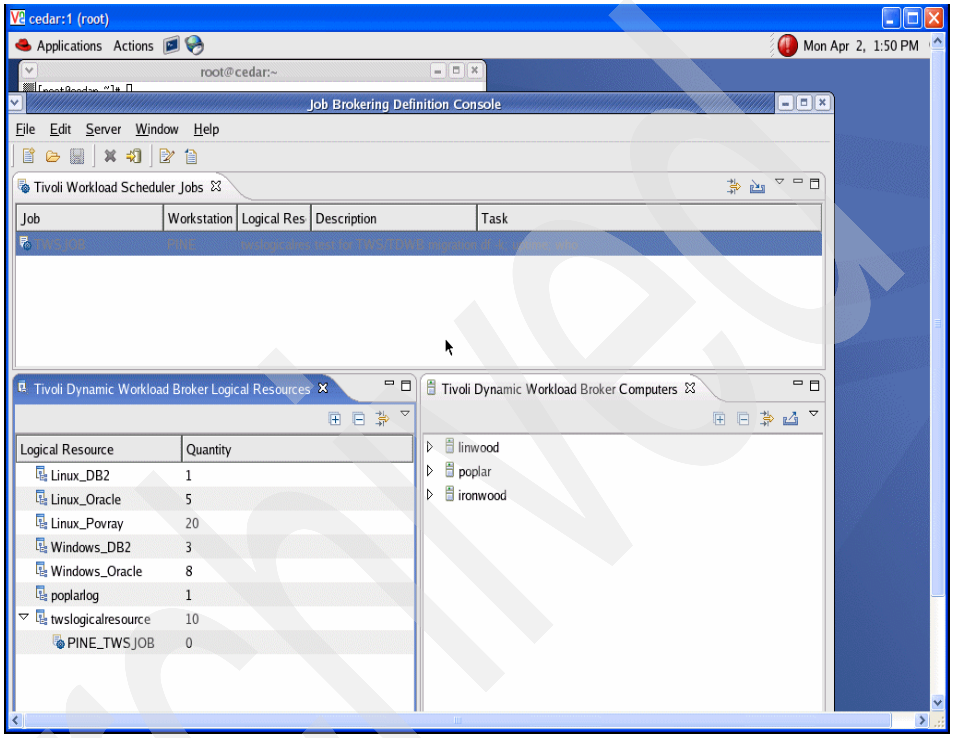
Job	Workstation	Logical Res	Description	Task
TWSJOB	PINE		test for TWS/TDWB migration	df -k; uptime; who

Logical Resource	Quantity
Linux_DB2	1
Linux_Oracle	5
Linux_Povray	20
Windows_DB2	3
Windows_Oracle	8
poplarlog	1
twslogicalresource	10

Figure 5-10 Both TWS jobs and Tivoli Dynamic Workload Broker Logical Resources should be displayed

6. Drag and drop the icon for **TWSJOB** onto the logical resource TWSLOGICALRESOURCE.

7. Click the logical resource to expand it. You should see PINE_TWSJOB. This is a local Tivoli Dynamic Workload Broker job definition that has been created by the import function. It will be exported from the Job Broker Definition Console machine to the Tivoli Dynamic Workload Broker server. See Figure 5-11.


cedar:1 (root)
Applications
Actions
Mon Apr 2, 1:50 PM
root@cedar:~
Job Brokering Definition Console
File
Edit
Server
Window
Help
Tivoli Workload Scheduler Jobs
Tivoli Dynamic Workload Broker Logical Resources
Tivoli Dynamic Workload Broker Computers

Job	Workstation	Logical Res	Description	Task
TWSJOB	PINE	twslogicalres	test for TWS/TDWB migration	df -k; uptime; who

Logical Resource	Quantity
Linux_DB2	1
Linux_Oracle	5
Linux_Povray	20
Windows_DB2	3
Windows_Oracle	8
poplarlog	1
twslogicalresource	10
PINE_TWSJOB	0

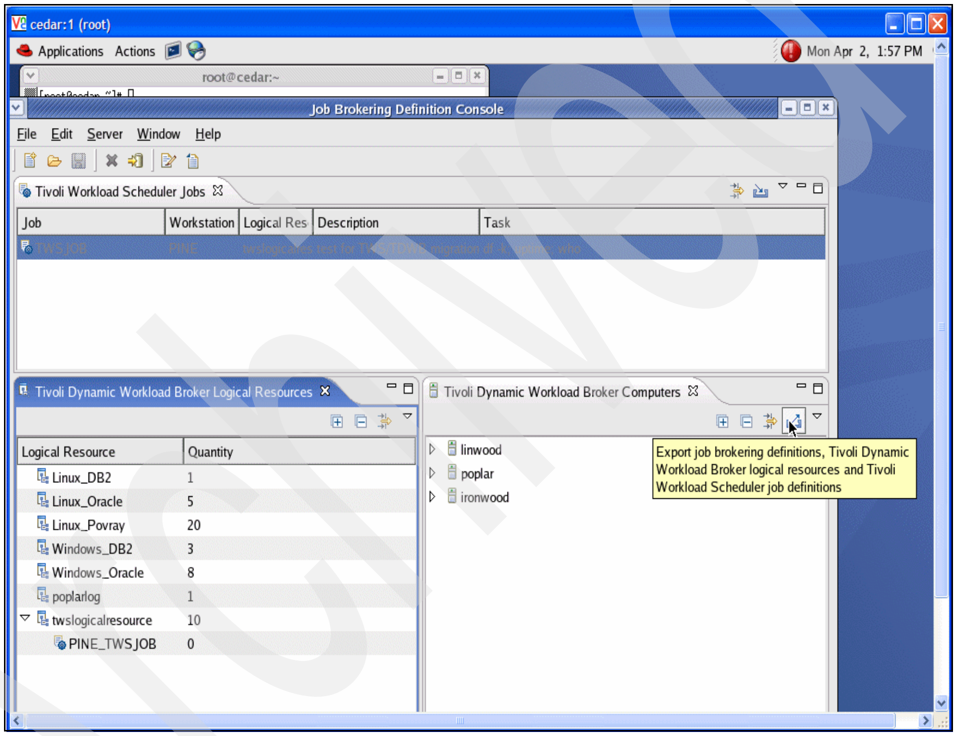
linwood
poplar
ironwood

Figure 5-11 PINE_TWSJOB is the Tivoli Dynamic Workload Broker job definition created by dragging and dropping TWSJOB onyo logical resource twslogicalresource

5.1.6 Export Tivoli Workload Scheduler job definitions and job streams to Tivoli Dynamic Workload Broker

Do the following to export Export Tivoli Workload Scheduler job definitions and job streams to the Tivoli Dynamic Workload Broker:

1. In the Tivoli Dynamic Workload Broker Computers window click the icon **Export job brokering definitions, Tivoli Dynamic Workload Broker logical resources, and Tivoli Workload Scheduler job definitions**, as shown in Figure 5-12.

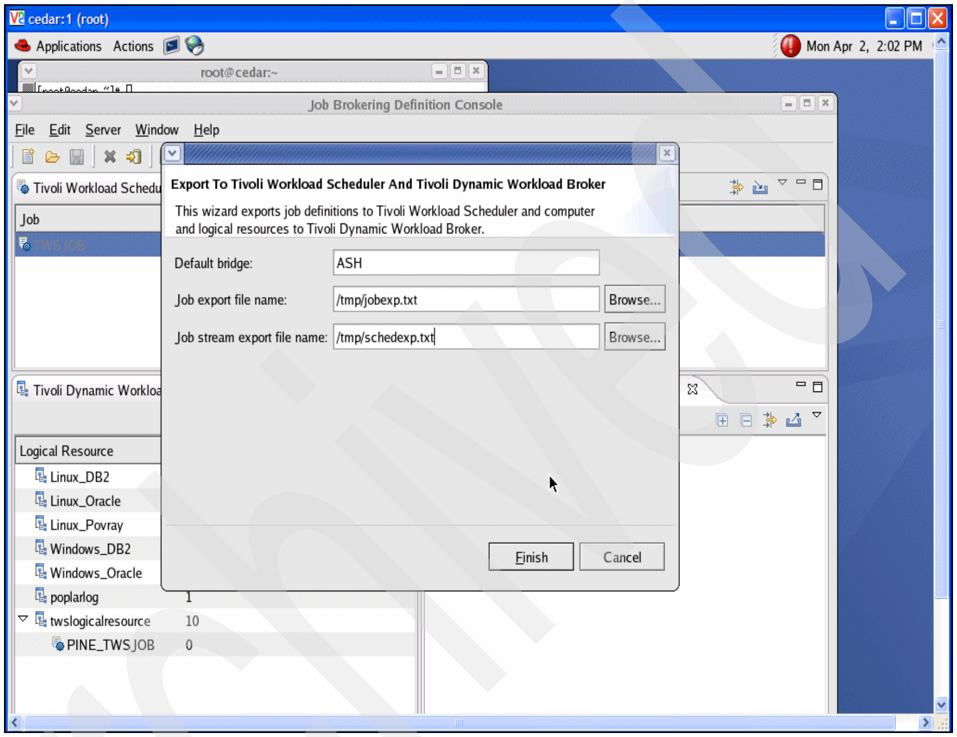

cedar:1 (root)
Applications Actions
Mon Apr 2, 1:57 PM
root@cedar:~
Job Brokering Definition Console
File Edit Server Window Help
Tivoli Workload Scheduler Jobs
Job Workstation Logical Res Description Task
Tivoli Dynamic Workload Broker Logical Resources
Tivoli Dynamic Workload Broker Computers
linwood
poplar
ironwood

Logical Resource	Quantity
Linux_DB2	1
Linux_Oracle	5
Linux_Povray	20
Windows_DB2	3
Windows_Oracle	8
poplarlog	1
twslogicalresource	10
PINE_TWSJOB	0

Export job brokering definitions, Tivoli Dynamic Workload Broker logical resources and Tivoli Workload Scheduler job definitions

Figure 5-12 Icon for Export job brokering definitions, Tivoli Dynamic Workload Broker logical resources, and Tivoli Workload Scheduler job definitions

2. In the Export to Tivoli Workload Scheduler and Tivoli Dynamic Workload Broker window (Figure 5-13) specify:
   – Default bridge: ASH
   – Job export file name: /tmp/jobexp.txt
   – Job stream export file name: /tmp/schedexp.txt



Export To Tivoli Workload Scheduler And Tivoli Dynamic Workload Broker

This wizard exports job definitions to Tivoli Workload Scheduler and computer and logical resources to Tivoli Dynamic Workload Broker.

Default bridge: ASH

Job export file name: /tmp/jobexp.txt [Browse...]

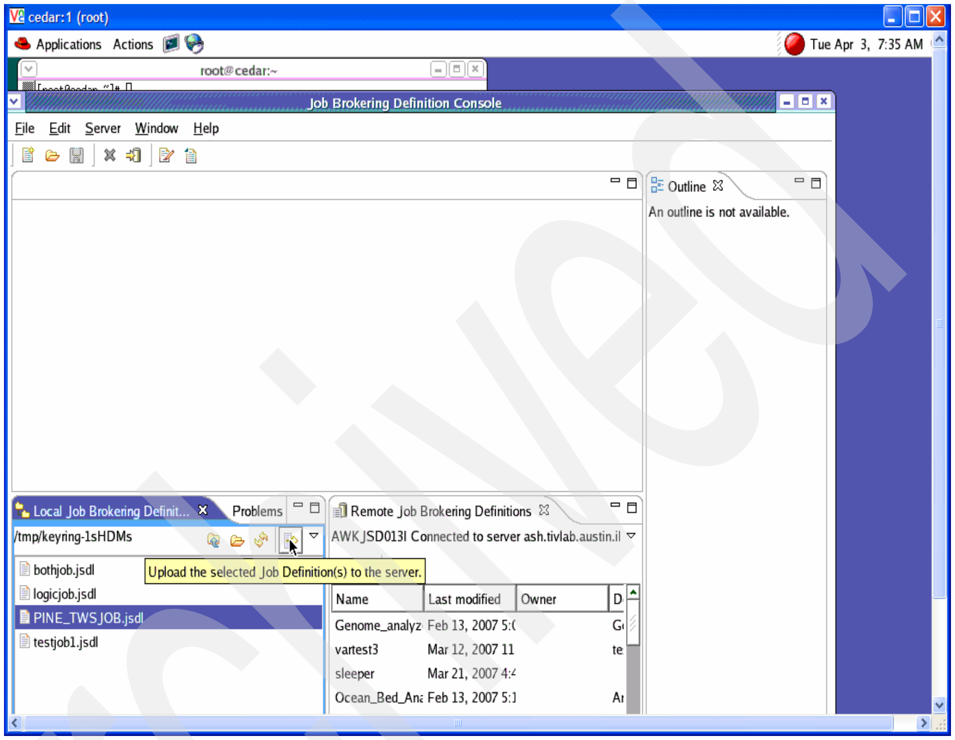
Job stream export file name: /tmp/schedexp.txt [Browse...]

[Finish] [Cancel]

Figure 5-13 Specify bridge and file names for Tivoli Workload Scheduler

3. Click **Finish** to create the new files /tmp/jobexp.txt and /tmp/schedexp.txt.

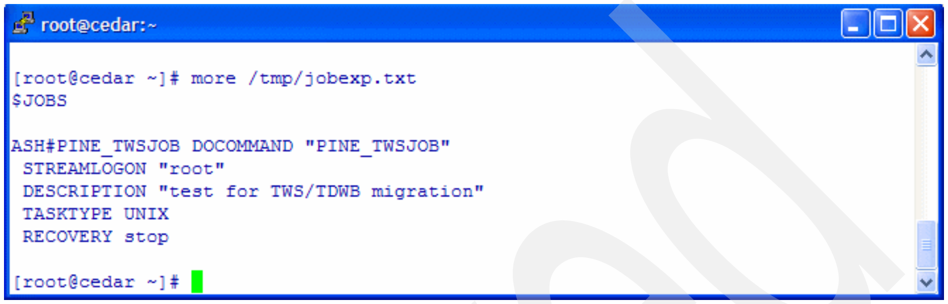
4. Select **Window** → **Open Perspective** → **Job Brokering Definition Editing**. In the Local Job Brokering Definitions window select **PINE_TWSJOB.jsdl** and click the **Upload** icon. This will put the new Tivoli Dynamic Workload Broker job definition into the Tivoli Dynamic Workload Broker server job repository database. See Figure 5-14.


cedar:1 (root)
Applications Actions
Tue Apr 3, 7:35 AM
root@cedar:~
Job Brokering Definition Console
File Edit Server Window Help
Outline
An outline is not available.
Local Job Brokering Definit...
Problems
Remote Job Brokering Definitions
/tmp/keyring-1sHDMs
Upload the selected Job Definition(s) to the server.
AWKJSD013I Connected to server ash.tivlab.austin.il
bothjob.jsdl
logicjob.jsdl
PINE_TWSJOB.jsdl
testjob1.jsdl

Name	Last modified	Owner	D
Genome_analyz	Feb 13, 2007 5:(		G
vartest3	Mar 12, 2007 11		te
sleeper	Mar 21, 2007 4:4		
Ocean_Bed_An:	Feb 13, 2007 5:1		A

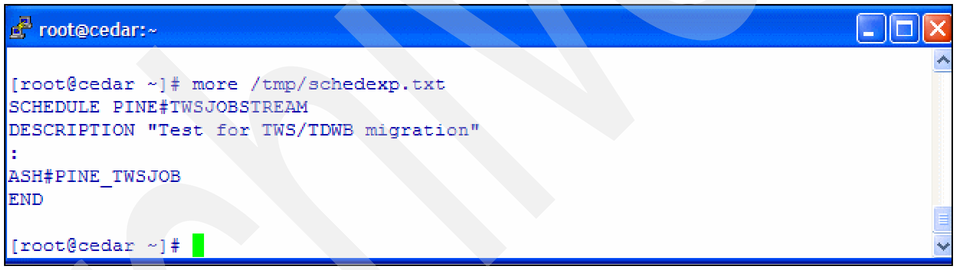
Figure 5-14 Upload new Tivoli Dynamic Workload Broker job definition PINE_TWSJOB to the Tivoli Dynamic Workload Broker server

5. Copy the files /tmp/jobexp.txt and /tmp/schedexp.txt to the Tivoli Workload Scheduler master domain manager. On the Tivoli Workload Scheduler master domain manager use the **`composer`** command to add the new Tivoli Workload Scheduler job definition and job stream to the Tivoli Workload Scheduler database. See Figure 5-15.



```
[root@cedar ~]# more /tmp/jobexp.txt
$JOBS

ASH#PINE_TWSJOB DOCOMMAND "PINE_TWSJOB"
 STREAMLOGON "root"
 DESCRIPTION "test for TWS/TDWB migration"
 TASKTYPE UNIX
 RECOVERY stop

[root@cedar ~]#
```

Figure 5-15 jobexp.txt fie contents for the new Tivoli Workload Scheduler job definition PINE_TWSJOB



```
[root@cedar ~]# more /tmp/schedexp.txt
SCHEDULE PINE#TWSJOBSTREAM
DESCRIPTION "Test for TWS/TDWB migration"
:
ASH#PINE_TWSJOB
END

[root@cedar ~]#
```

Figure 5-16 schedexp.txt file contents for the updated Tivoli Workload Scheduler job stream TWSJOBSTREAM

6. You can now schedule or submit the updated Tivoli Workload Scheduler job stream TWSJOBSTREAM or the new Tivoli Workload Scheduler job definition PINE_TWSJOB in Tivoli Workload Scheduler. The result will be Tivoli Dynamic Workload Broker running the Tivoli Dynamic Workload Broker job definition PINE_TWSJOB on a computer in the logical resource TWSLOGICALRESOURCE.

5.2 Job Submission Description Language reference

The Job Submission Description Language (JSDL) is a language for describing the job requirements for submission to resources. The JSDL language contains a vocabulary and normative XML schema that facilitate the expression of those requirements as a set of XML elements.

To create and edit JSDL files, you can use the Job Brokering Definition Console. The JSDL files are saved in the Job Repository as job definitions and become available for submission. JSDL files adhere to the XML syntax and semantics as defined in the JSDL schema. The JSDL file is arranged in a hierarchical structure where the jobDefinition element is the root element. The jobDefinition element contains all the elements that describe the job, the resource requirements, and preferences attributes and scheduling directives.

The pseudo schema definition looks like Example 5-1.

Example 5-1 Pseudo schema definition

```
< jobDefinition >
   <annotation ... />?
   <category>... />*
   <variables ... />?
   <application ... />
   <resources ... />?
   <relatedResources ... />*
   <optimization ... >?
   <scheduling ...>?
</jobDefinition>
```

5.2.1 Category element

This element is used to categorize the job. One job may have multiple categories, for example, IT_Employees, DB_Job, SalarySystem, and Batch. Currently this element is useful only in the case of the Enterprise Workload Manager integration. See the JSDL snippet in Example 5-2.

Example 5-2 Category element

```
<jsdl:category>IT_Employees</jsdl:category>
  <jsdl:category>DB_Job</jsdl:category>
  <jsdl:category>SalarySystem</jsdl:category>
  <jsdl:category>Batch</jsdl:category>
```

5.2.2 Variable element

The variable element has only one set of variables. It is used to define the default values of some or all variables used in the JSDL. See Example 5-3.

Example 5-3 Variable element

```
<jsdl:variables>
    <jsdl:stringVariable description="Address where to send salaries"
name="IT_mail_address">5th Avenue  - Austin</jsdl:stringVariable>
    <jsdl:uintVariable description="Number of working days for the
current week" name="Number_of_working_days">5</jsdl:uintVariable>
    <jsdl:uintVariable description="this is the number of the client
connections used through the the db client"
name="Salary_db_client">10</jsdl:uintVariable>
  </jsdl:variables>
Used in …
 <jsdle:arguments>
        <jsdle:value>${IT_mail_address}</jsdle:value>
        <jsdle:value>${Number_of_working_days}</jsdle:value>
      </jsdle:arguments>
And…
 <jsdl:logicalResource name="Salary" quantity="${Salary_db_client}"
subType="DBClientConnection"/>
```

There are three types of variables: string, double, integer. Variable values can be overridden at submit time with variable values passed from the Web UI or CLI. Only XXXXVariableExpressionType elements or attributes can reference variables using ${*var_name*}, where *var_name* may or may not be defined in the variables element. See Example 5-4.

Example 5-4 Referencing variables

```
<xsd:simpleType name="UnsignedIntVariableExpressionType">
<xsd:union>
<xsd:simpleType>
<xsd:restriction base='xsd:unsignedInt' />
</xsd:simpleType>
<xsd:simpleType>
<xsd:restriction base='xsd:string'>
<xsd:pattern value="[\n\r\t ]*($\{[a-zA-Z_]+[0-9a-zA-Z_\.\-]*\})[\n\r\t
]*" />
</xsd:restriction>
</xsd:simpleType>
</xsd:union>
</xsd:simpleType>
```

```
<xsd:complexType name="LogicalResourceRequirementType">
   <xsd:attribute name="quantity"
   type="jsdl:UnsignedIntVariableExpressionType" use="optional" />
   <xsd:attribute name="name"
   type="jsdl:StringVariableExpressionType" use="optional" />
   <xsd:attribute name="subType"
   type="jsdl:StringVariableExpressionType" use="optional" />
</xsd:complexType>
```

5.2.3 Application element

This is a mandatory element. It defines the type of the job and includes what has to be executed and the actual parameters to be used. The type is specified in the name attribute and only two values are allowed for now:

- "executable"
- "j2ee"

Example 5-5 is an example JSDL snippet.

Example 5-5 Application element

```
<jsdl:application name="executable">
<jsdl:application name="j2ee">
```

Depending on the type specified two different elements may be specified.

5.2.4 Execution element

Executable element allows the specifications for native jobs (that is, executables and scripts). Scripts may be embedded in this element. See Example 5-6.

Example 5-6 Execution element

```
<xsd:complexType name="ExecutableType">
<xsd:sequence>
<xsd:element name="arguments" type="jsdle:ArgumentsType" minOccurs="0"
maxOccurs="1" />
<xsd:element name="environment" type="jsdle:EnvironmentType"
minOccurs="0" maxOccurs="1" />
<xsd:element name="script" type="xsd:string" minOccurs="0"
maxOccurs="1" />
<xsd:element name="credential" type="jsdle:CredentialType"
minOccurs="0" maxOccurs="1" />
```

```
</xsd:sequence>
<xsd:attribute name="path" type="jsdl:StringVariableExpressionType"
use="optional" />
<xsd:attribute name="error" type="jsdl:StringVariableExpressionType"
use="optional" />
<xsd:attribute name="input" type="jsdl:StringVariableExpressionType"
use="optional" />
<xsd:attribute name="output" type="jsdl:StringVariableExpressionType"
use="optional" />
<xsd:attribute name="workingDirectory"
type="jsdl:StringVariableExpressionType" use="optional" />
</xsd:complexType>
JSDL snippet for executable specification:
 <jsdl:application name="executable">
    <jsdle:executable error="salary.err" input="salary.in"
output="salary.out" path="calculate_salary.sh"
workingDirectory="/home/salaryuser/">
      <jsdle:arguments>
        <jsdle:value>${IT_mail_address}</jsdle:value>
        <jsdle:value>${Number_of_working_days}</jsdle:value>
      </jsdle:arguments>
      <jsdle:credential>
        <jsdle:userName>salaryuser</jsdle:userName>
      </jsdle:credential>
    </jsdle:executable>
  </jsdl:application>
```

Execution element - embedded scripts

Example 5-7 is an example of JSDL snippets for embedded script specification.

Example 5-7 JSDL snippets for embedded script specification

```
Windows
<jsdle:executable output="outwin.log" workingDirectory="${wd}">
      <jsdle:arguments>
        <jsdle:value>${string}</jsdle:value>
        <jsdle:value>${filename}</jsdle:value>
      </jsdle:arguments>
      <jsdle:script>FIND &quot;%1&quot; &quot;%2&quot;</jsdle:script>
</jsdle:executable>
Unix/Linux
 <jsdle:executable error="script.err" output="script.out"
workingDirectory="/tmp/TDWB">
      <jsdle:arguments>
```

```
        <jsdle:value>${start}</jsdle:value>
        <jsdle:value>${end}</jsdle:value>
        <jsdle:value>${filename}</jsdle:value>
      </jsdle:arguments>
      <jsdle:script>#!/bin/sh
    if [ $# -eq 3 ]; then
if [ -e $3 ]; then
        tail +$1 $3 | head -n$2
         else
        echo &quot;$0: Error opening file $3&quot;
    exit 2
fi
    else
        echo &quot;Missing arguments!&quot;
    fi</jsdle:script>
</jsdle:executable>
```

5.2.5 J2EE element

The J2EE element allows the specifications for J2EE jobs, for example, Java Message Services (JMS) messages and Enterprise Java Beans (EJB) calls. It may contain three elements:

- "invoker": This can be direct or indirect.
  - Indirect means that the EJB call or JMS message is executed through the WAS scheduler. After the scheduler call the agent sends the EXECUTING status to the server and then starts polling the scheduler for updating the job status to successfully or unsuccessfully completed.
  - Direct means that the Tivoli Dynamic Workload Broker agent directly calls the EJB *process* method or directly sends the JMS message. The agent job's thread remains blocked until the operation completes. The EXECUTING status is not sent to the server.
- "jms" or "ejb": These elements are mutually exclusive.
  - JMSelement allows the specification of the target JMS queue and the message to be sent.
  - The EJB element allows the specification of the JNDI home of the EJB to be called.
- Credential: This element contains the credentials to be used when invoking the EJB or sending the JMS message.

J2EE example for a Direct EJB job

See the JSDL schema snippet example in Example 5-8.

Example 5-8 JSDL schema snippet for direct EJB job

```
<xsd:complexType name="J2EEType">
   <xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="invoker"
type="jsdlj:InvokerType"/>
<xsd:choice maxOccurs="1" minOccurs="1">
<xsd:element maxOccurs="1" minOccurs="0" name="jms"
type="jsdlj:JMSActionType"/>
<xsd:element maxOccurs="1" minOccurs="0" name="ejb"
type="jsdlj:EJBActionType"/>
</xsd:choice>
<xsd:element maxOccurs="1" minOccurs="0" name="credential"
type="jsdlj:CredentialType"/>
</xsd:sequence>
</xsd:complexType>
```

Example 5-9 shows a JSDL XML snippet for a Direct EJB job.

Example 5-9 JSDL XML snippet for direct EJB job

```
   <jsdl:application name="j2ee">
       <jsdlj:j2ee>
         <jsdlj:invoker type="Direct"/>
         <jsdlj:ejb>

   <jsdlj:jndiHome>ejb/test/scheduler/MyTest100BeanHome</jsdlj:jndiHome
   >
         </jsdlj:ejb>
       </jsdlj:j2ee>
     </jsdl:application>
```

This job causes the Tivoli Dynamic Workload Broker agent to directly call the EJB ejb/test/scheduler/MyTest100Bean *process* method. The EJB must be already installed in the target WAS and must implement the TaskHandler interface.

J2EE example for an Indirect EJB job

Example 5-10 shows a JSDL XML snippet for an Indirect EJB job.

Example 5-10 J2EE example - indirect EJB job

```
   <jsdl:application name="j2ee">
       <jsdlj:j2ee>
         <jsdlj:invoker type="Indirect"/>
         <jsdlj:ejb>

   <jsdlj:jndiHome>ejb/test/scheduler/MyTest100BeanHome</jsdlj:jndiHome
   >
         </jsdlj:ejb>
       </jsdlj:j2ee>
     </jsdl:application>
```

This job causes the Tivoli Dynamic Workload Broker agent to indirectly call the EJB ejb/test/scheduler/MyTest100Bean process method through the WebSphere Application Server Scheduler. The EJB must be already installed in the target WebSphere Application Server and must implement the TaskHandler interface. The WebSphere Application Server scheduler must be already configured in the target WAS and its address defined in the configuration property file of the agent.

J2EE example for a Direct JMS job

Example 5-11 shows a JSDL XML snippet for a Direct EJB job.

Example 5-11 JSDL XML snippet for a direct EJB job

```
<jsdl:application name="j2ee">
    <jsdlj:j2ee>
      <jsdlj:invoker type="Direct"/>
      <jsdlj:jms>
        <jsdlj:connFactory>jms/MyCF</jsdlj:connFactory>
        <jsdlj:destination>jms/MyQueue</jsdlj:destination>
        <jsdlj:message>Hello Paolo</jsdlj:message>
      </jsdlj:jms>
    </jsdlj:j2ee>
  </jsdl:application>
```

This job causes the Tivoli Dynamic Workload Broker agent to directly send the message `Hello Paolo` to jms/MyQueue address.

J2EE example for an indirect JMS job

Example 5-12 shows a JSDL XML snippet for an indirect JMS job.

Example 5-12   JSDL XML snippet for an indirect JMS job

```
<jsdl:application name="j2ee">
    <jsdlj:j2ee>
      <jsdlj:invoker type="Indirect"/>
      <jsdlj:jms>
        <jsdlj:connFactory>jms/MyCF</jsdlj:connFactory>
        <jsdlj:destination>jms/MyQueue</jsdlj:destination>
        <jsdlj:message>Hello Paolo</jsdlj:message>
      </jsdlj:jms>
    </jsdlj:j2ee>
  </jsdl:application>
```

This job causes the Tivoli Dynamic Workload Broker agent to indirectly send the message “Hello Paolo” to jms/MyQueue address. The WebSphere Application Server scheduler must already be configured in the target WebSphere Application Server and its address defined in the configuration property file of the agent.

5.2.6 Resource element

Resource element IDs are used for specifications for resource requirements that must be matched on target computer systems in order for a job to be dispatched to that system.

All resource requirements below must be matched when using the AND statement. Some of the requirements are themselves a list of requirements that may be considered when using the OR statement (that is, at least one of them must be matched). A resource may contain the following elements:

- candidateHosts: This is a list of possible host names to be matched.
- candidateCPUs: This is a list of possible CPU characteristics to be matched.
- physicalMemory: The range of free physical memory to be matched.
- virtualMemory: The range of free virtual memory to be matched.
- candidateOperatingSystems: This is a list of possible OS characteristics to be matched.
- fileSystem: This is the exact list of file system characteristics to be matched.
- logicalResource. This is the exact list of logical resource characteristics to be matched.

- Group: This is the exact list of groups to which the target computer must be associated.

> **Note:** Only resouce groups containing computers can be used. Logical resource groups containing logical resources can only defined as a related resource.

- Properties: This is the AND/OR combination of computer system attribute range values that must be matched on target.
- Allocation: This is the list of computer system attribute exact values that must logically reserved for this job.
- Relationship: This is the list of relationships with resources defined in the relatedResources element.

Resource element - CandidateHosts

This is an optional element. It allows the specification of a list of possible host names. The hosts specified are in OR. At least one of them must be matched by the operating system resource contained in the target computer system resource.

Each hostName element can be a reference to a variable (that is, ${my_host}). The hostName element supports wildcards.

Example 5-13 shows an example.

Example 5-13 Resource element - CandidateHosts

```
<xsd:complexType name="CandidateHostsRequirementType">
<xsd:complexContent>
<xsd:sequence>
<xsd:element name="hostName"
type="jsdl:StringVariableExpressionType" minOccurs="1"
maxOccurs="unbounded" />
</xsd:sequence>
</xsd:complexContent>
</xsd:complexType>
```

This is an optional element. It allows the specification of a list of possible CPU characteristic combinations. The characteristic combinations specified are in OR. At least one of them must be matched by the target computer system resource.

- The “architecture” attribute can have one of the following values: “powerpc”, “powerpc_64”, “x86”, “x86_64”, “s390”, or “s390x”.
- The “quantity” attribute indicates the number of processors required.

- The “speed” element indicates the CPU speed range in MHz required.

Example 5-14 shows a JSDL schema snippet for resource element - CandidateCPUs.

Example 5-14 JSDL schema snippet

```
<xsd:complexType name="CandidateCPUsRequirementType">
<xsd:complexContent>
<xsd:sequence>
<xsd:element name="cpu"
type="jsdl:CPURequirementType" minOccurs="1"
maxOccurs="unbounded" />
</xsd:sequence>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="CPURequirementType">
<xsd:complexContent>
<xsd:sequence>
<xsd:element name="speed"
type="jsdl:NumericRangeType" minOccurs="0" maxOccurs="1" />
</xsd:sequence>
<xsd:attribute name="architecture"
type="jsdl:ProcessorArchitectureEnumeration"
use="optional" />
<xsd:attribute name="quantity" type="xsd:unsignedInt"
use="optional" />
</xsd:complexContent>
</xsd:complexType>
```

Example 5-15 shows a JSDL XML snippet for resource element - CandidateCPUs.

Example 5-15 JSDL XML snippet for Resource element - CandidateCPUs

```
   <jsdl:candidateCpus>
         <jsdl:cpu architecture="x86" quantity="2">
           <jsdl:speed>
             <jsdl:range>
               <jsdl:minimum>1000.0</jsdl:minimum>
               <jsdl:maximum>2000.0</jsdl:maximum>
             </jsdl:range>
           </jsdl:speed>
         </jsdl:cpu>
         <jsdl:cpu architecture="x86" quantity="1">
```

```
          <jsdl:speed>
            <jsdl:range>
              <jsdl:minimum>2000.0</jsdl:minimum>
              <jsdl:maximum>4000.0</jsdl:maximum>
            </jsdl:range>
          </jsdl:speed>
        </jsdl:cpu>
      </jsdl:candidateCpus>
```

Resource element - candidateOperationSystem

This is an optional element. It allows the specification of a list of possible operating system characteristic combinations. The characteristic combinations are matched in OR. At least one of them must be matched By The Operating System Resource Contained In The Target Computer System Resource.

The “type” attribute can have one of the following values: “AIX”, “LINUX”, “Windows 2000”, “Windows XP”, or “Windows 2003”. The “version” attribute indicates the version in dotted notation majVer.minVer.updVer.

Example 5-16 is an example JSDL schema snippet for resource element - candidateOperationSystem.

Example 5-16 JSDL schema snippet for resource element - candidateOperationSystem

```
<xsd:attribute name="version" type="xsd:string“ use="optional" />
<xsd:attribute name="type“ type="jsdl:OperatingSystemTypeEnumeration"
use="required" />
</xsd:complexContent>
</xsd:complexType>
   <xsd:complexType name="OperatingSystemsRequirementType">
   <xsd:complexContent>
   <xsd:sequence>
   <xsd:element name="operatingSystem“
   type="jsdl:OperatingSystemRequirementType" minOccurs="1“
   maxOccurs="unbounded" />
   </xsd:sequence>
   </xsd:complexContent>
   </xsd:complexType>
<xsd:complexType name="OperatingSystemRequirementType">
<xsd:complexContent>
```

Example 5-17 is an example JSDL schema XML snippet for resource element - candidateOperationSystem.

Example 5-17 JSDL schema XML snippet for resource element - candidateOperationSystem

```
   <jsdl:candidateOperatingSystems>
         <jsdl:operatingSystem type="LINUX" version="3.0"/>
         <jsdl:operatingSystem type="AIX" version="5.2"/>
    </jsdl:candidateOperatingSystems>
```

Resource element - fileSystem

This is an optional element. It allows the specification of a list of required file system characteristic combinations. The characteristic combinations are matched in AND. All must be matched by the file system resources contained in the target computer system resource.

The “type” attribute can have one of the following values: "Unknown”, “No Root Directory”, “Removable Disk”, “Local Disk”, “Remote Drive”, “CD-ROM”, or “RAM Disk”.

The “mountPoint” attribute indicates the mount point, and it can be a reference to a variable (that is, ${temp_fs}). The mountPoint attribute supports wildcards. See Example 5-18.

Example 5-18 Resource element - fileSystem JSDL schema snippet

```
<xsd:complexType name="FileSystemRequirementType">
<xsd:complexContent>
<xsd:sequence>
<xsd:element name="diskSpace"
type="jsdl:NumericRangeType" minOccurs="0" maxOccurs="1" />
</xsd:sequence>
<xsd:attribute name="type“ type="jsdl:FileSystemTypeEnumeration"
use="optional“ default="Local Disk" />
<xsd:attribute name="mountPoint"
type="jsdl:StringVariableExpressionType" use="optional" />
</xsd:complexContent>
</xsd:complexType>
```

Both file systems must exist on the target computer.

Example 5-19 is another example of a JSDL XML snippet for resource element - candidateOperationSystem.

Example 5-19 JSDL schema snippet for resource element - candidateOperationSystem

```
<jsdl:fileSystem mountPoint="/home/salaryuser" type="No Root
Directory">
      <jsdl:diskSpace>
        <jsdl:range>
          <jsdl:minimum>100000.0</jsdl:minimum>
        </jsdl:range>
      </jsdl:diskSpace>
    </jsdl:fileSystem>
    <jsdl:fileSystem mountPoint="/temp" type="No Root Directory">
      <jsdl:diskSpace>
        <jsdl:range>
          <jsdl:minimum>1.0E7</jsdl:minimum>
        </jsdl:range>
      </jsdl:diskSpace>
    </jsdl:fileSystem>
```

Resource element - memory

This is an optional element. It allows the specification of a range of physical or virtual free memory. If specified, the ranges must be matched by the operating system resource contained in the target computer system resource.

Example 5-20 is an example JSDL schema snippet.

Example 5-20 Resource element - memory

```
<xsd:element name="physicalMemory"
   type="jsdl:NumericRangeType" minOccurs="0" maxOccurs="1" />
<xsd:element name="virtualMemory"
   type="jsdl:NumericRangeType" minOccurs="0" maxOccurs="1" />
JSDL XML snippet
<jsdl:virtualMemory>
      <jsdl:range>
        <jsdl:minimum>100000.0</jsdl:minimum>
      </jsdl:range>
</jsdl:virtualMemory>
```

Resource element - logical resource

This is an optional element. It allows the specification of a list of required logical resource characteristic combinations. The characteristic combinations are matched in AND. All must be matched by the logical resources “AssociatedWith” the target computer system resource.

All attributes can be a reference to a variable (that is, ${MyAppl}). The name and subType attributes support wildcards. If the “quantity” attribute is specified then that value is also reserved in the allocation process. In other words, the quantity does not only define a requirement but also a quantity to be reserved.

In order to be matched, the logical resources with those characteristics have to exist in the Tivoli Dynamic Workload Broker resource repository. They must also have the “AssociateWith” relationship with the target resource, matching all the other requierements expressed in the resources element.

Example 5-21 is an example JSDL schema snippet.

Example 5-21 JSDL schema snippet for Resource element - logical resource

```
<xsd:element name="logicalResource“
type="jsdl:LogicalResourceRequirementType" minOccurs="0“
maxOccurs="unbounded" />
<xsd:complexType name="LogicalResourceRequirementType">
<xsd:attribute name="quantity"
type="jsdl:UnsignedIntVariableExpressionType" use="optional" />
<xsd:attribute name="name"
type="jsdl:StringVariableExpressionType" use="optional" />
<xsd:attribute name="subType"
type="jsdl:StringVariableExpressionType" use="optional" />
</xsd:complexType>
```

Example 5-22 is an example JSDL XML snippet.

Example 5-22 JSDL XML snippet for Resource element - logical resource

```
<jsdl:logicalResource name="Salary" quantity="${Salary_db_client}"
subType="DBClientConnection"/>
```

Resource element - group

This is an optional element. It allows the specification of a list of required groups. The groups are matched in AND. The target computer system must be in all the groups specified.

The name attribute can be a reference to a variable (that is, ${MyAppl}). The name attribute supports wildcards. In order to be matched, the groups have to exist in the Tivoli Dynamic Workload Broker resource repository, and they must also contain only target computer systems, not logical resources.

Example 5-23 is an example JSDL schema snippet.

Example 5-23 JSDL schema snippet for Resource element - group

```
<xsd:element name="group" type="jsdl:GroupRequirementType"
           minOccurs="0" maxOccurs="unbounded" />

<xsd:complexType name="GroupRequirementType">
<xsd:complexContent>
<xsd:attribute name="name"
type="jsdl:StringVariableExpressionType" use="required" />
</xsd:complexContent>
</xsd:complexType>
```

Example 5-24 is an example JSDL XML snippet.

Example 5-24 JSDL XML snippet for Resource element - group

```
<jsdl:group name="DBServersGroup"/>
 <jsdl:group name="HumanResourceGroup"/>
```

Resource element - properties

This is an optional element. It allows the specification of a list of combined AND/OR requirements on the attributes of the computer system.

Example 5-25 is an example JSDL schema snippet.

Example 5-25 JSDL schema snippet for Resource element - properties

```
<xsd:element name="properties“ type="jsdl:RequirementCompositorType"
minOccurs="0“ maxOccurs="1" />
<xsd:complexType name="RequirementCompositorType">
   <xsd:group ref="jsdl:CompositorContent" />
</xsd:complexType>
<xsd:group name="CompositorContent">
<xsd:sequence>
<xsd:element name="and“ type="jsdl:RequirementCompositorType"
minOccurs="0“
   maxOccurs="unbounded" />
<xsd:element name="or" type="jsdl:RequirementCompositorType“
minOccurs="0"
   maxOccurs="unbounded" />
<xsd:element name="requirement"
type="jsdl:RequirementType"minOccurs="0"
   maxOccurs="unbounded" />
</xsd:sequence>
```

```
</xsd:group>
<xsd:complexType name="RequirementType">
<xsd:complexContent>
<xsd:extension base="jsdl:StringRangeType">
   <xsd:attribute name="propertyName" type="xsd:QName“ use="required"
/>
</xsd:extension>
</xsd:complexContent>
```

Example 5-26 is an example JSDL XML snippet.

Example 5-26   JSDL XML snippet for Resource element - properties

```
   <jsdl:properties>
         <jsdl:or>
           <jsdl:requirement propertyName="CPUUtilization">
             <jsdl:range>
               <jsdl:maximum>20</jsdl:maximum>
             </jsdl:range>
           </jsdl:requirement>
           <jsdl:requirement propertyName="ProcessingSpeed">
             <jsdl:range>
               <jsdl:minimum>4000</jsdl:minimum>
             </jsdl:range>
           </jsdl:requirement>
         </jsdl:or>
         <jsdl:requirement propertyName="ProcessorType">
           <jsdl:exact>x86_64</jsdl:exact>
         </jsdl:requirement>
       </jsdl:properties>
```

Resource element - allocation

This is an optional element. It allows the specification of the logical reservation of the specified quantity of consumable attributes of the computer system.

The element can be a reference to a variable (that is, ${MyQuantity}).

Example 5-27 shows an example of a JSDL schema snippet.

Example 5-27 JSDL schema snippet for Resource element - allocation

```
<xsd:complexType name="AllocationRequirementType">
<xsd:simpleContent>
<xsd:extension base="jsdl:DoubleVariableExpressionType">
   <xsd:attribute name="propertyName" type="xsd:QName"
      use="required" />
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
```

Example 5-28 shows an example of a JSDL XML snippet.

Example 5-28 JSDL XML snippet for Resource element - allocation

```
<jsdl:allocation propertyName="NumOfProcessors">1.0</jsdl:allocation>
```

Resource element - relationship

This is an optional element. It allows the specification of the relationships with resource requirements specified in the relatedResource elements.

Example 5-29 shows an example of a JSDL schema snippet.

Example 5-29 JSDL schema snippet for Resource element - relationship

```
<xsd:complexType name="RelationshipRequirementType">
<xsd:complexContent>
<xsd:extension base="jsdl:ExtensibleElementsType">
<xsd:attribute name="type" type="xsd:QName"
use="optional" />
<xsd:attribute name="source" type="xsd:IDREF"
use="optional" />
<xsd:attribute name="target" type="xsd:IDREF"
use="optional" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

Example 5-30 shows an example of a JSDL XML snippet.

Example 5-30 JSDL XML snippet for Resource element - relationship

```
<jsdl:relationship target="My_OS" type="Contains"/>
<jsdl:relationship target="MY_LG"/>
<jsdl:relationship target="MY_FS" type="Contains"/>
…
<jsdl:relatedResources id="My_OS" type="OperatingSystem">
…
<jsdl:relatedResources id="MY_FS" type="FileSystem">
…
<jsdl:relatedResources id="MY_LG" type="LogicalResource">
```

5.2.7 Related resources element

A related resources element allows the specifications of resource requirements for resources that may have or not have a relationship with the target computer system. They are operating system, file system, network system, and logical resource.

All resource requirements below must be matched in AND.

This element could be used when:

- The user wants to specify more complex AND/OR requirements on OS, NS, FS, or LR.
- The user wants to specify requirements on all OS, NS, FS, or LR attributes.
- The user wants to use a resource as Global with or without relationship.

It may contain the following elements:

- logicalResource: This is the exact list of logical resource characteristics to be matched.
- Group: This is the exact list of groups to which the related resource type must be associated.
- Properties: This is the AND/OR combination of related resource type attribute range values that must be matched on target.
- Allocation: This is the list of related resource type attribute exact values that must logically reserved for this job.
- Relationship: This is the list of relationships with resources defined in other relatedResources elements.

It contains the following attributes:

- type: The type of the related resource. Allowed values are ComputerSystem, OperatingSystem, FileSystem, NetworkSystem, and LogicalResource.
- id: This is a JSDL unique identifier used only within the JSDL to allow relationShips to target any relatedResource.

Example 5-31 shows an example of a JSDL schema snippet.

Example 5-31 JSDL schema snippet for related resources element

```
<xsd:group name="generalRequirements">
<xsd:sequence>
<xsd:element name="properties“ type="jsdl:RequirementCompositorType"
minOccurs="0"
maxOccurs="1" />
<xsd:element name="group" type="jsdl:GroupRequirementType“
minOccurs="0" maxOccurs="unbounded" />
<xsd:element name="allocation“ type="jsdl:AllocationRequirementType"
minOccurs="0"
maxOccurs="unbounded" />
<xsd:element name="relationship“
type="jsdl:RelationshipRequirementType" minOccurs="0“
maxOccurs="unbounded" />
</xsd:sequence>
</xsd:group>
<xsd:complexType name="RelatedResourceType">
<xsd:complexContent>
<xsd:extension base="jsdl:ExtensibleElementsType">
<xsd:sequence>
   <xsd:group ref="jsdl:generalRequirements" />
</xsd:sequence>
<xsd:attribute name="type" type="xsd:QName“ use="optional" />
<xsd:attribute name="id" type="xsd:ID" use="required" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

Example 5-32 shows an example of a JSDL XML snippet.

Example 5-32 JSDL XML snippet for related resources element

```
<jsdl:relatedResources id="MyLR" type="LogicalResource">
<jsdl:properties>
<jsdl:requirement propertyName="Subtype">
   <jsdl:exact>Pippo</jsdl:exact>
</jsdl:requirement>
</jsdl:properties>
<jsdl:group name="MyCriticalLG" />
<jsdl:allocation propertyName="Quantity">1</jsdl:allocation>
</jsdl:relatedResources>
```

Resource element versus related resource element

Example 5-33 and Example 5-34 compare the resource element versus the related resource element.

Example 5-33 JSDL XML snippets

```
<jsdl:resources >
<jsdl:candidateOperatingSystems>
<jsdl:operatingSystem type="Windows XP" version="5.1" />
   </jsdl:candidateOperatingSystems>
</jsdl:resources >
```

This is equivalent to Example 5-34.

Example 5-34 JSDL XML snippets

```
<jsdl:resources >
   <jsdl:relationship type="Contains" target="MyOS" />
</jsdl:resources >
<jsdl:relatedResources id="MyOS" type="OperatingSystem">
<jsdl:properties>
   <jsdl:and>
<jsdl:requirement propertyName="OperatingSystemVersion">
   <jsdl:exact>5.1</jsdl:exact>
</jsdl:requirement>
<jsdl:requirement propertyName="OperatingSystemType">
   <jsdl:exact>Windows XP</jsdl:exact>
</jsdl:requirement>
</jsdl:and>
</jsdl:properties>
</jsdl:relatedResources>
```

5.2.8 Optimization element

The optimization element allows the specifications of the resource selection policy. It may contain the following elements:

- EWLM: The EWLM provides the resource weights.
- objective: The TDWB calculates the resource preferences based on actual resource conditions.
- Name: This allows the TDWB to know which policy has been specified. Allowed values are:
  - JPT_JSDLOptimizationPolicyType
  - JPT_EWLMType

Example 5-35 shows an example of a JSDL schema snippet.

Example 5-35 Optimization element

```
<xsd:complexType name="OptimizationType">
<xsd:complexContent>
   <xsd:extension base="jsdl:ExtensibleElementsType">
   <xsd:choice minOccurs="1">
   <xsd:element name="ewlm" type="jsdl:EWLMType" />
   <xsd:element name="objective"
   type="jsdl:PropertyObjectiveType" />
   </xsd:choice>
   <xsd:attribute name="name" type="xsd:NCName"
   use="optional" default="JPT_JSDLOptimizationPolicyType" />
   </xsd:extension>
   </xsd:complexContent>
   </xsd:complexType>
```

Optimization element - objective

This element defines the resource preference criteria based on resource attribute values. It contains the following attributes:

- propertyObjective: the operation to be applied on the resource attribute specified in “resourcePropertyName”
  - Minimize and maximize apply to all optimizable attributes and give more preference to the resource with minimum or maximum attribute values.
  - MinimizeUtilization and MaximizeUtilization apply only to consumable attributes (that is, those that can be specified in the allocation”element. Those operations give more preference to the resources that have the consumable attribute less or more used (that is, allocated). These

operations are very useful to concentrate or distribute the load over a set of resources.

- resourceType: the type of the resource that has the attribute “resourcePropertyName”. Allowed values are ComputerSystem, OperatingSystem, NetworkSystem, FileSystem, and LogicalResource.
- resourcePropertyName: the attribute to which the operation must be applied
  - Depends on resource type.
  - Only optimizable attributes can be specified (that is, integer, double, float, and so on). JSDLEditor allows this filtering.

Example 5-36 shows an example of a JSDL schema snippet.

Example 5-36   Optimization element - objective

```
<xsd:complexType name="PropertyObjectiveType">
<xsd:attribute name="propertyObjective" default="minimize"
use="optional">
<xsd:simpleType>
<xsd:restriction base="xsd:NCName">
<xsd:enumeration value="minimize" />
<xsd:enumeration value="maximize" />
<xsd:enumeration value="minimizeUtilization" />
<xsd:enumeration value="maximizeUtilization" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="resourceType" type="xsd:QName"
use="required" />
<xsd:attribute name="resourcePropertyName" type="xsd:QName"
use="required" />
</xsd:complexType>
```

Example 5-37 shows an example of a JSDL XML snippet.

Example 5-37 Optimization element - objective

```
<jsdl:optimization>
<jsdl:objective resourceType="ComputerSystem"
resourcePropertyName="CPUUtilization" />
</jsdl:optimization>
On consumable attributes:
<jsdl:optimization>
<jsdl:objective objective="maximizeUtilization"
resourceType="LogicalResource"
resourcePropertyName="Quantity" />
</jsdl:optimization>
```

5.2.9 Scheduling element

The scheduling element allows the specifications of scheduling directives for the scheduler system (for example, Tivoli Dynamic Workload Broker). It may contain the following elements:

- Priority: the job priority
- Estimated duration: the expected job duration
- Maximum resource waiting time: the maximum time the TDWB has to wait before determining that there are no resources matching the requirements
- Recovery actions: the list of actions (for now only TPM actions) to be executed when no capable resources are found

Example 5-38 shows an example of a JSDL schema snippet.

Example 5-38 Scheduling element

```
<xsd:complexType name="SchedulingType">
<xsd:complexContent>
<xsd:extension base="jsdl:ExtensibleElementsType">
<xsd:sequence>
<xsd:element name="recoveryActions"
type="jsdl:RecoveryActionList" minOccurs="0" maxOccurs="1" />
<xsd:element name="maximumResourceWaitingTime"
type="xsd:duration" minOccurs="0" maxOccurs="1" />
<xsd:element name="estimatedDuration"
type="xsd:duration" minOccurs="0" maxOccurs="1" />
<xsd:element maxOccurs="1" minOccurs="0"
name="priority" type="jsdl:PriorityType" />
```

```
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

Example 5-38 on page 215 shows an example of a JSDL schema snippet.

Example 5-39 Scheduling element

```
<jsdl:scheduling>
<jsdl:recoveryActions>
</jsdl:recoveryActions>
<jsdl:maximumResourceWaitingTime>P0DT1M</jsdl:maximumResourceWaiting
Time> (One Minute: Very important job. If there are no resources it
will fail)
<jsdl:estimatedDuration>P1D</jsdl:estimatedDuration> (one day - long
running job)
<jsdl:priority>100</jsdl:priority> (high Priority)
</jsdl:scheduling>
```

Scheduling - recovery actions

This allows the specifications of recovery actions to be executed when no capable resources are found (that is, maximum resource waiting time is expired).

The JSDL schema can be extended by other schemas. It defines the common attributes and elements for any action. An unlimited number of actions can be specified. The Tivoli Dynamic Workload Broker executes all actions sequentially and under the condition that the action returns an exit code = 0.

The RecoveryActionList element is a set of unlimited RecoveryActions. Each extensible RecoveryActionType is made of the following attributes:

- name: This name defines the type of the action. For now only the “tpmaction” value is allowed and must match the name of the recovery action plugin defined in the Resource Advisor configuration.
- additionalTimeOnCompletion: The time to wait after the action completes before completing the sequence or invoking the next action.
- maximumExecutionTime: The maximum time to wait for an action to complete. After its expiration the TDWB stops the action sequence.

Example 5-40 shows an example of a JSDL schema snippet.

Example 5-40 Scheduling - recovery actions

```
<xsd:complexType name="RecoveryActionType">
<xsd:complexContent>
<xsd:extension base="jsdl:ExtensibleElementsType">
<xsd:attribute name="name" type="xsd:NCName" use="required" />
<xsd:attribute name="additionalTimeOnCompletion" type="xsd:duration"
use="optional" default="P0DT0S" />
<xsd:attribute name="maximumExecutionTime" type="xsd:duration"
use="optional" default="P0DT0S" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="RecoveryActionList">
<xsd:complexContent>
<xsd:extension base="jsdl:ExtensibleElementsType">
<xsd:sequence>
<xsd:element name="action"
type="jsdl:RecoveryActionType" minOccurs="1" maxOccurs="unbounded" />
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

Scheduling - TPM Extension

This allows the specifications of Tivoli Provisioning Manager recovery actions. The JSDL schema can be extended by other schemas. It defines the common attributes and elements for any action. The TPMAction schema is an extension to the JSDL schema. It defines the TPMActionType.

- Parameters: the arguments to be passed to the workflow
- Credential: the credential to be used when connecting to TPM
- Tpmaddress: the address of the TPM server
- workFlow: the name of the workflow

Note: The default credential and address are defined in TPMConfig.properties. These can be overridden per job.

Example 5-41 shows an example of a JSDL schema snippet.

Example 5-41 TPMAction schema snippet

```
<xsd:complexType name="ParametersType">
<xsd:sequence>
<xsd:element name="parameter" minOccurs="1"
maxOccurs="unbounded">
<xsd:complexType>
<xsd:simpleContent>
<xsd:extension base="jsdl:StringVariableExpressionType">
<xsd:attribute name="name"
type="xsd:string" use="required"/>
</xsd:extension>
</xsd:simpleContent>
   </xsd:complexType>
   </xsd:element>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="TPMAddressType">
<xsd:attribute name="host" type="xsd:string" use="optional"/>
<xsd:attribute name="port" type="xsd:unsignedInt" use="optional"/>
</xsd:complexType>
<xsd:element name="tpmaction" type="jsdltpm:TPMActionType"/>
<xsd:complexType name="TPMActionType">
<xsd:sequence>
<xsd:element name="parameters" type="jsdltpm:ParametersType"
minOccurs="0" maxOccurs="1" />
<xsd:element name="credential" type="jsdl:CredentialType" minOccurs="0"
maxOccurs="1"/>
<xsd:element name="tpmaddress" type="jsdltpm:TPMAddressType"
minOccurs="0" maxOccurs="1"/>
</xsd:sequence>
   <xsd:attribute name="workFlow"
type="jsdl:StringVariableExpressionType" use="required" />
</xsd:complexType>
```

5.3 Tivoli Dynamic Workload Broker user authorization and authentication

Global security should be enabled to protect unauthorized users from accessing and changing WebSphere Application Server configurations and Tivoli Dynamic

Workload Broker Server definitions, configuration, and job and resource repositories.

When global security is not enabled, any user can access the WebSphere Application Server Administrative Console and connect to the Tivoli Dynamic Workload Broker console using the Job Brokering Definition Console, command line, or Tivoli Dynamic Workload Broker Web Console.

Once a user or group is mapped to a security role on the Tivoli Dynamic Workload Broker server, that authority carries through for the user when accessing the Tivoli Dynamic Workload Broker server via the Job Brokering Definition Console, via the command-line interface, or via the Tivoli Dynamic Workload Broker Web Console.

Enabling global security and mapping security roles to users or groups is done via the WebSphere Application Server Administrative Console.

Access the WebSphere Application Server Administrative Console via a Web browser:

```
http://tdwb_server:9060/ibm/console
```

The first time you log into the WebSphere Administrative Console no user ID is required because security is not enabled.

When you have enabled global security, you must always provide a user name and password for the connection to the Tivoli Dynamic Workload Broker server for the following interfaces:

- Tivoli Job Brokering Definition Console

  A valid user name and password must be supplied when defining a server connection to the Tivoli Dynamic Workload Broker server.

- Tivoli Dynamic Workload Broker Web Console

  This is also known as the Integrated Solutions Console. A valid user name and password must be supplied when configuring a server connection to the Tivoli Dynamic Workload Broker Server.

- Command-line interface

  The user and password required to connect to the Tivoli Dynamic Workload Broker can be set in the *TDWB_Server_Install_Directory/config/*CLIConfig.properties file or when issuing a command-line interface command. More information about command-line interface commands can be found in 5.4, “Command-line interface” on page 232.

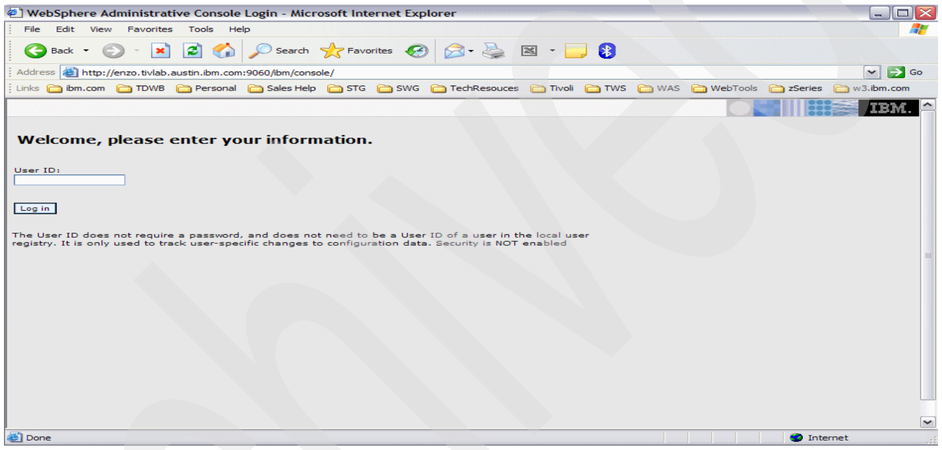
- Tivoli Workload Scheduler agent

  The user and password required to connect to the Tivoli Dynamic Workload Broker can set in the *TDWB_Server_Install_Directory/config/*TWSAgent.properties file.

5.3.1 Enabling global security

Perform the following steps to enable global security:

1. Access the WebSphere Application Server Administrative Console via a Web browser (`http://tdwb_server:9060/ibm/console`), as shown in Figure 5-17.


WebSphere Administrative Console Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://enzo.tivlab.austin.ibm.com:9060/ibm/console/
Go
Links ibm.com TDWB Personal Sales Help STG SWG TechResouces Tivoli TWS WAS WebTools zSeries w3.ibm.com
IBM.
Done
Internet

Welcome, please enter your information.

User ID:

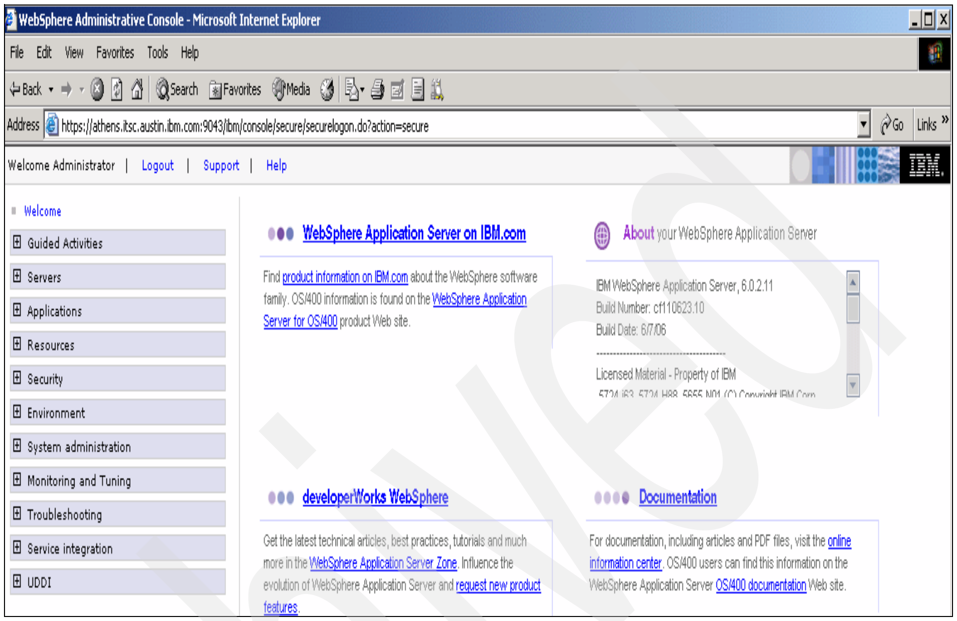
Log in

The User ID does not require a password, and does not need to be a User ID of a user in the local user registry. It is only used to track user-specific changes to configuration data. Security is NOT enabled

Figure 5-17 WebSphere Application Server Administrative Console login screen

Note: The first time you log into the WebSphere Administrative Console no user ID is required because security is not enabled.

2. From the WebSphere Application Server Administrative Console Screen click the plus sign (+) next to Security on the list of Welcome tasks on the left side of the screen, as shown in Figure 5-18.


WebSphere Administrative Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address https://athens.itsc.austin.ibm.com:9043/ibm/console/secure/securelogon.do?action=secure
Go Links
Welcome Administrator | Logout | Support | Help
Welcome
Guided Activities
Servers
Applications
Resources
Security
Environment
System administration
Monitoring and Tuning
Troubleshooting
Service integration
UDDI
WebSphere Application Server on IBM.com
About your WebSphere Application Server
developerWorks WebSphere
Documentation

Find product information on IBM.com about the WebSphere software family. OS/400 information is found on the WebSphere Application Server for OS/400 product Web site.

IBM WebSphere Application Server, 6.0.2.11
Build Number: cf110623.10
Build Date: 6/7/06

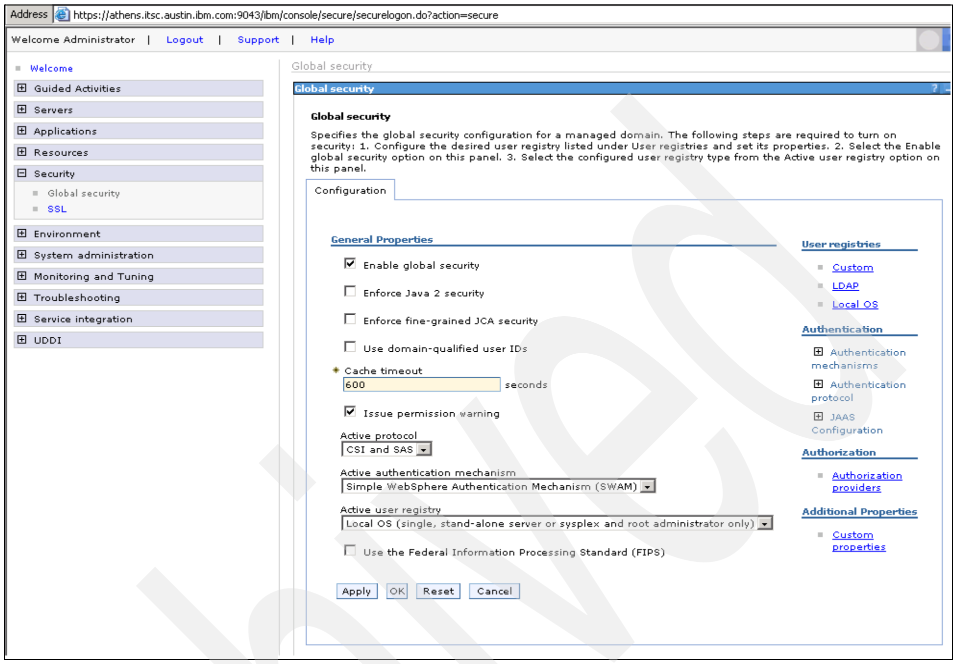
Licensed Material - Property of IBM
5724-I63, 5724-H88, 5655-N01 (C) Copyright IBM Corp.

Get the latest technical articles, best practices, tutorials and much more in the WebSphere Application Server Zone. Influence the evolution of WebSphere Application Server and request new product features.

For documentation, including articles and PDF files, visit the online information center. OS/400 users can find this information on the WebSphere Application Server OS/400 documentation Web site.

Figure 5-18 WebSphere Application Server Administrative Console Welcome page

3. Click **Global security**, select **Enable global security**, and deselect **Enforce Java 2 security**, as shown in Figure 5-19, and click **Apply**.


Address https://athens.itsc.austin.ibm.com:9043/ibm/console/secure/securelogon.do?action=secure

Welcome Administrator | Logout | Support | Help

Global security

Global security

Specifies the global security configuration for a managed domain. The following steps are required to turn on security: 1. Configure the desired user registry listed under User registries and set its properties. 2. Select the Enable global security option on this panel. 3. Select the configured user registry type from the Active user registry option on this panel.

Configuration

General Properties

[✓] Enable global security

[ ] Enforce Java 2 security

[ ] Enforce fine-grained JCA security

[ ] Use domain-qualified user IDs

* Cache timeout: 600 seconds

[✓] Issue permission warning

Active protocol: CSI and SAS

Active authentication mechanism: Simple WebSphere Authentication Mechanism (SWAM)

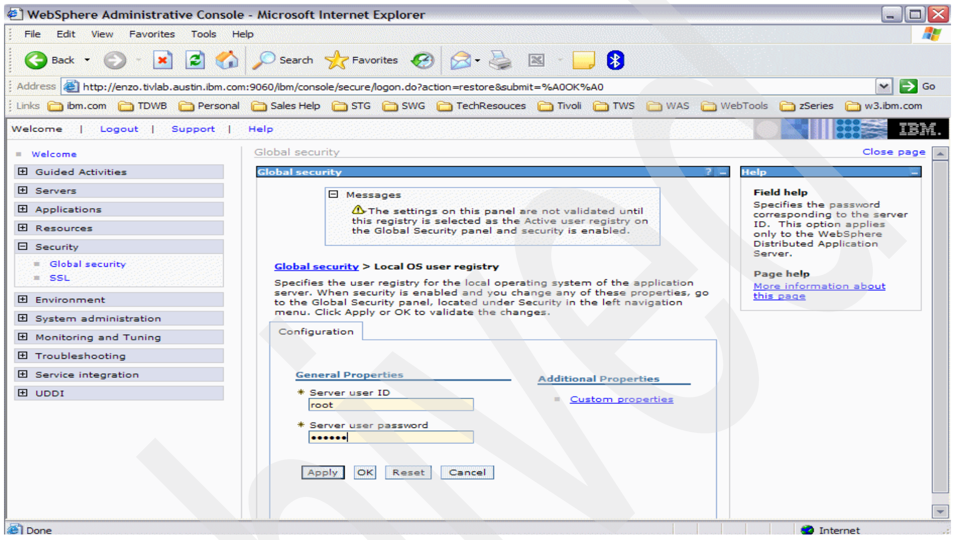
Active user registry: Local OS (single, stand-alone server or sysplex and root administrator only)

[ ] Use the Federal Information Processing Standard (FIPS)

Figure 5-19 Enable global security

Important: If you do not deselect the Enforce Java 2 security option Tivoli Dynamic Workload Broker will not work.

4. Configure the WebSphere Application Server Administrative Console user for global security, as shown in Figure 5-20. The user must be a valid user for the local operating system of the WebSphere Application Server. This will be the user ID and password that will be required to access the WebSphere Application Server Administrative Console and make future configuration changes. Click **Apply** and **Save**.


WebSphere Administrative Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://enzo.tivlab.austin.ibm.com:9060/ibm/console/secure/logon.do?action=restore&submit=%A0OK%A0 Go
Links ibm.com TDWB Personal Sales Help STG SWG TechResouces Tivoli TWS WAS WebTools zSeries w3.ibm.com
Welcome | Logout | Support | Help
IBM.
Welcome
Guided Activities
Servers
Applications
Resources
Security
Global security
SSL
Environment
System administration
Monitoring and Tuning
Troubleshooting
Service integration
UDDI
Global security
Close page
Done
Internet

Global security

Messages

⚠ The settings on this panel are not validated until this registry is selected as the Active user registry on the Global Security panel and security is enabled.

Global security > **Local OS user registry**

Specifies the user registry for the local operating system of the application server. When security is enabled and you change any of these properties, go to the Global Security panel, located under Security in the left navigation menu. Click Apply or OK to validate the changes.

Configuration

General Properties

* Server user ID
root

* Server user password
••••••

Additional Properties

Custom properties

Apply OK Reset Cancel

Help

Field help

Specifies the password corresponding to the server ID. This option applies only to the WebSphere Distributed Application Server.

Page help

More information about this page

Figure 5-20 Configuring the WebSphere Application Server Administrative Console user

5.3.2 Tivoli Dynamic Workload Broker security roles

When you have enabled global security on the WebSphere Application Server, all of the users defined in the server user registry are authenticated users and are given the administrator role. You should map security roles to users or groups of users in order to better control your Tivoli Dynamic Workload Broker environment.

The roles discussed in this section are available in the WebSphere Application Server for the Tivoli Dynamic Workload Broker:

WSClient

This is a superset of all of the other roles. Use all authenticated or specify the users or groups that you assigned for the other roles. We recommend that you leave this as *all authenticated* so that other applications that might want to connect via Web services calls can do so.

Important: Do *not* select the Everyone check box for this role. If you select Everyone, Tivoli Workload Broker will not work.

Administrator

Users or groups of users with this security role are considered Tivoli Dynamic Workload Broker superusers and have full authorization (configurator, developer, operator, and submitter). However, users or groups with this security role cannot add or mofiy users for the Tivoli Dynamic Workload Broker Web Console.

Operator

Users or groups of users with this security role can:

- Define, edit, and delete logical resources.
- Define, edit, and delete resource groups.
- Modify server connections.
- View job definitions.
- Submit jobs.
- Track jobs and computers.

Users or groups of users with this security role cannot:

- Add or modify job definitions.
- Install new Tivoli Workload Broker Agents.
- Add or modify users for the Tivoli Dynamic Workload Broker Web Console.

Submitter

Users or groups of users with this security role can:

- Submit jobs using the command-line interface.
- Manage any jobs that were submitted by this user via the command-line interface.
- Submit .jsdl files as Tivoli Dynamic Workload Broker jobs, but these jobs are *not* added to the job repository.

Users or groups of users with this security role cannot connect to the Tivoli Dynamic Workload Broker server with the Tivoli Job Brokering Definition Console or Tivoli Dynamic Workload Broker Web Interface.

Configurator

Users or groups of users with this security role can:

- Install new Tivoli Workload Broker Agents.
- Define, edit, and delete logical resources.
- Define, edit, and delete resource groups.
- Track computers.
- Modify server connections.

Users or groups of users with this security role cannot:

- Add, modify, view, track, or delete job definitions.
- Add or modify users for the Tivoli Dynamic Workload Broker Web Console.

Developer

Users or groups of users with this security role can:

- Define, edit, and delete logical resources.
- Define, edit, and delete resource groups.
- Modify server connections.
- View job definitions.
- Submit jobs.
- Track jobs and computers.

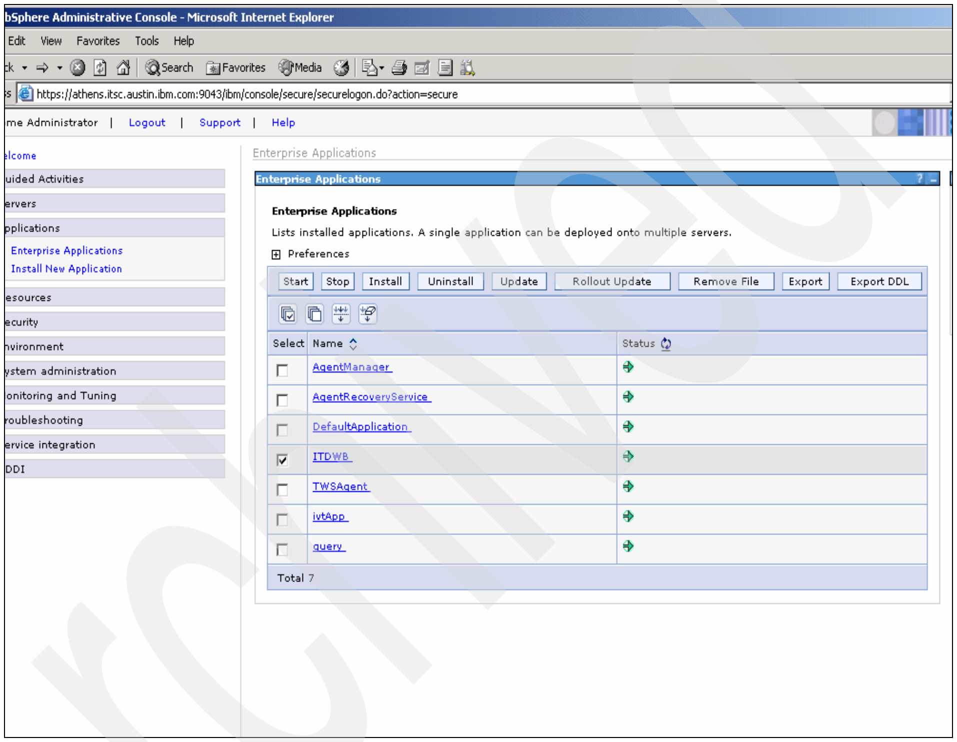
Users or groups of users with this security role cannot:

- Add or modify job definitions.
- Install new Tivoli Workload Broker Agents.
- Add or modify users for the Tivoli Dynamic Workload Broker Web Console.

5.3.3 Mapping security roles to users or groups

Perform the following steps to map security roles to users or groups of users:

1. From the WebSphere Application Server Administrative Console Welcome page, expand **Applications** on the Welcome Task List, click **Enterprise Applications**, and select **ITDWB**, as shown on Figure 5-21. Then click **Map security roles to users/groups**.

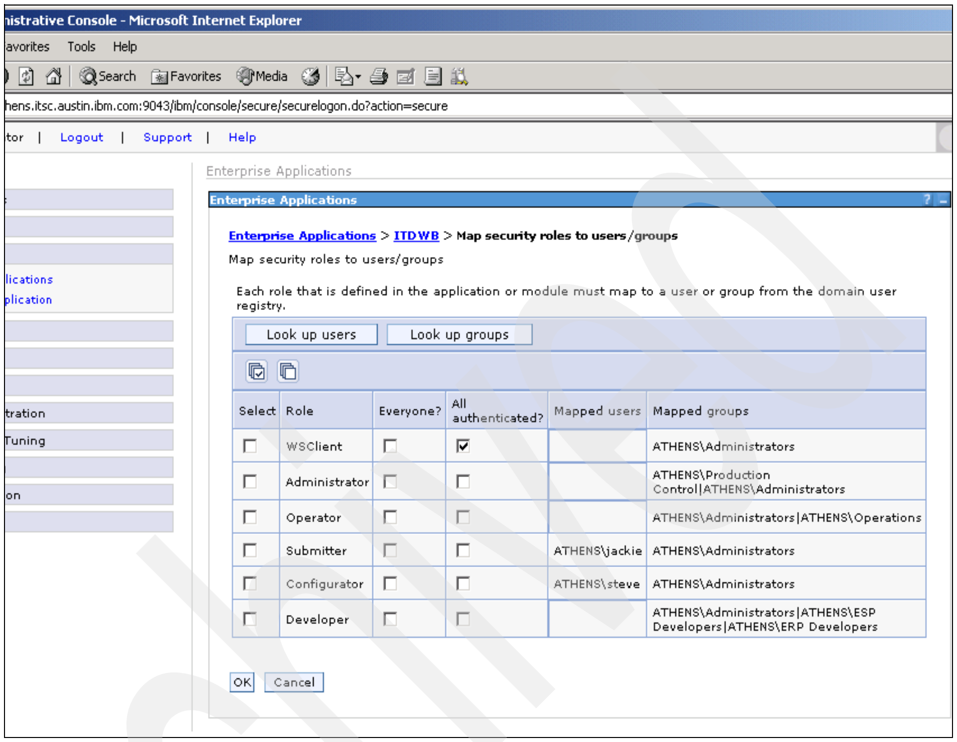


Select	Name	Status
☐	AgentManager	➔
☐	AgentRecoveryService	➔
☐	DefaultApplication	➔
☑	ITDWB	➔
☐	TWSAgent	➔
☐	ivtApp	➔
☐	query	➔

Total 7

Figure 5-21 Selecting Tivoli Dynamic Workload Broker Application

2. Assign users or groups to each security role, as shown in Figure 5-22. *All Authenticated Users* is specified for WSClient, so the mapping of the ATHENS\Administrator group is ignored.


Enterprise Applications > ITDWB > Map security roles to users/groups

Map security roles to users/groups

Each role that is defined in the application or module must map to a user or group from the domain user registry.

Select	Role	Everyone?	All authenticated?	Mapped users	Mapped groups
☐	WSClient	☐	☑		ATHENS\Administrators
☐	Administrator	☐	☐		ATHENS\Production Control|ATHENS\Administrators
☐	Operator	☐	☐		ATHENS\Administrators|ATHENS\Operations
☐	Submitter	☐	☐	ATHENS\jackie	ATHENS\Administrators
☐	Configurator	☐	☐	ATHENS\steve	ATHENS\Administrators
☐	Developer	☐	☐		ATHENS\Administrators|ATHENS\ESP Developers|ATHENS\ERP Developers

Figure 5-22 Tivoli Dynamic Workload Broker security roles

For a definition of each role see 5.3.2, “Tivoli Dynamic Workload Broker security roles” on page 223.

5.3.4 Manage users for Tivoli Dynamic Workload Broker Web Console

The Tivoli Dynamic Workload Broker Web Console is hosted by the Integrated Solutions Console. It enforces the authentication mechanism by default, has its own authentication mechanism, and maintains its own credential vault. The Integrated Solutions Console keeps all of the defined credentials in the embedded Cloudscape database.

The Integrated Solutions Console (ISC) does not use any operating system authentication, nor does it integrate with any external LDAP. Each user that wants to access any application running within the Integrated Solutions Console (such as the Tivoli Dynamic Workload Broker Web Console) is authenticated against the Integrated Solutions Console credential vault.

The user that wants to use the Tivoli Dynamic Workload Broker Web Console to access the Tivoli Dynamic Workload Broker server sees only one visible authentication step — he must provide the user ID and password defined in the Integrated Solutions Console.

The logon credentials are not the same on the Integrated Solutions Console and on the operating system (or LDAP) that uses the WebSphere Application Server (hosting the Tivoli Dynamic Workload Broker server). *Both credential vaults are not automatically synchronized in any way*. Due to this, a mapping among the logon definitions stored in the Integrated Solutions Console and authority that is used by WebSphere Application Server (hosting the Tivoli Dynamic Workload Broker server) must be done. The mapping is performed on the Integrated Solutions Console side.

The Integrated Solutions Console for the Tivoli Dynamic Workload Broker is preconfigured for assigning users to Tivoli Dynamic Workload Broker views.

Note: Authorization for each user is determined by the user name and password defined with the server connection, *not* by the user name and password used when logging into the Tivoli Dynamic Workload Broker Web Console.

5.3.5 Add users to Tivoli Dynamic Workload Broker Web Console roles

To add users to Tivoli Dynamic Workload Broker Web Console roles:

1. Log into the Tivoli Dynamic Workload as iscadmin.
2. Click **Console Settings** → **User and Group Management**.

3. Click **Search** to display available groups, as shown in Figure 5-23.

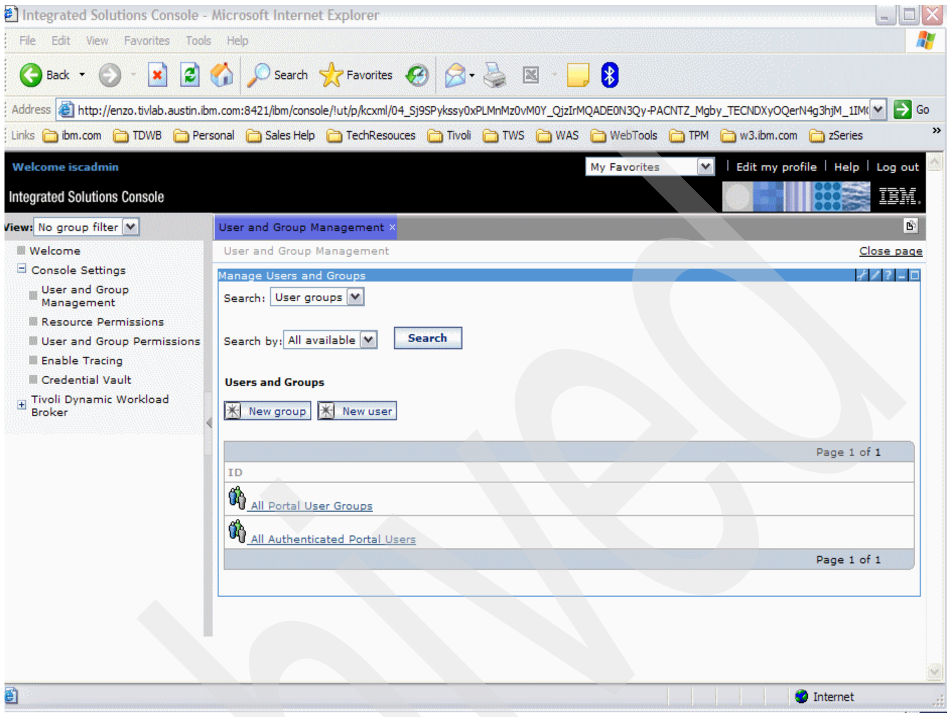

Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://enzo.tivlab.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/04_Sj9SPykssy0xPLMnMz0vM0Y_QjzIrMQADE0N3Qy-PACNTZ_Mgby_TECNDXyOQerN4g3hjM_1IM
Go
Links ibm.com TDWB Personal Sales Help TechResouces Tivoli TWS WAS WebTools TPM w3.ibm.com zSeries
Welcome iscadmin
My Favorites | Edit my profile | Help | Log out
Integrated Solutions Console
View: No group filter
Welcome
Console Settings
User and Group Management
Resource Permissions
User and Group Permissions
Enable Tracing
Credential Vault
Tivoli Dynamic Workload Broker
User and Group Management
User and Group Management
Close page
Manage Users and Groups
Search: User groups
Search by: All available
Search
Users and Groups
New group
New user
Page 1 of 1
ID
All Portal User Groups
All Authenticated Portal Users
Page 1 of 1
Internet

Figure 5-23 Searching for ISC groups

Figure 5-24 shows the ISC groups.

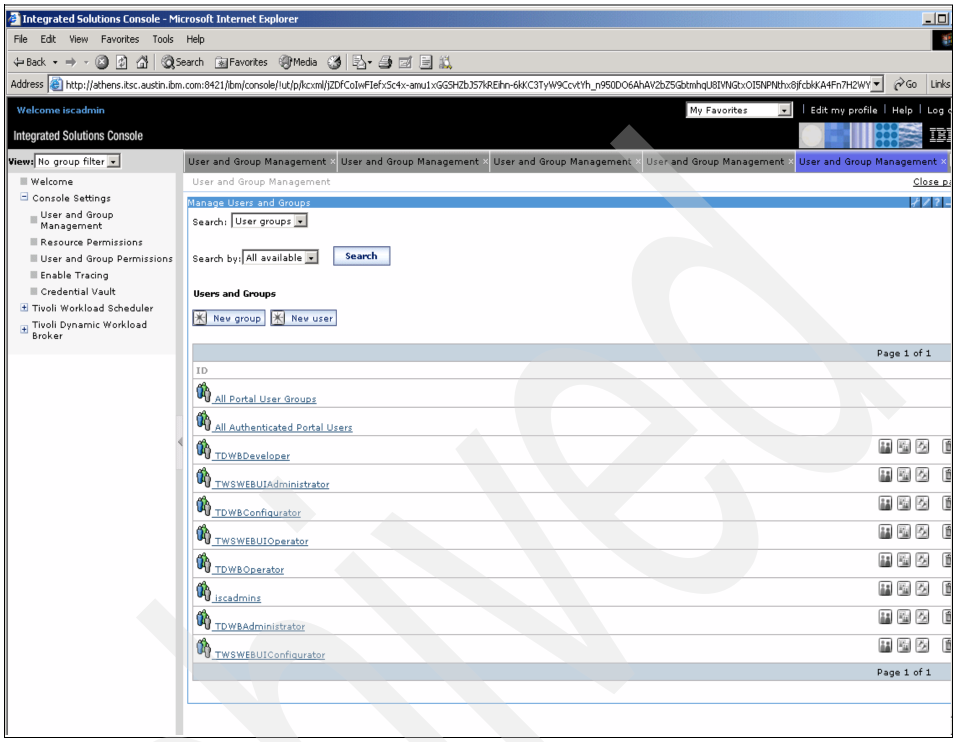
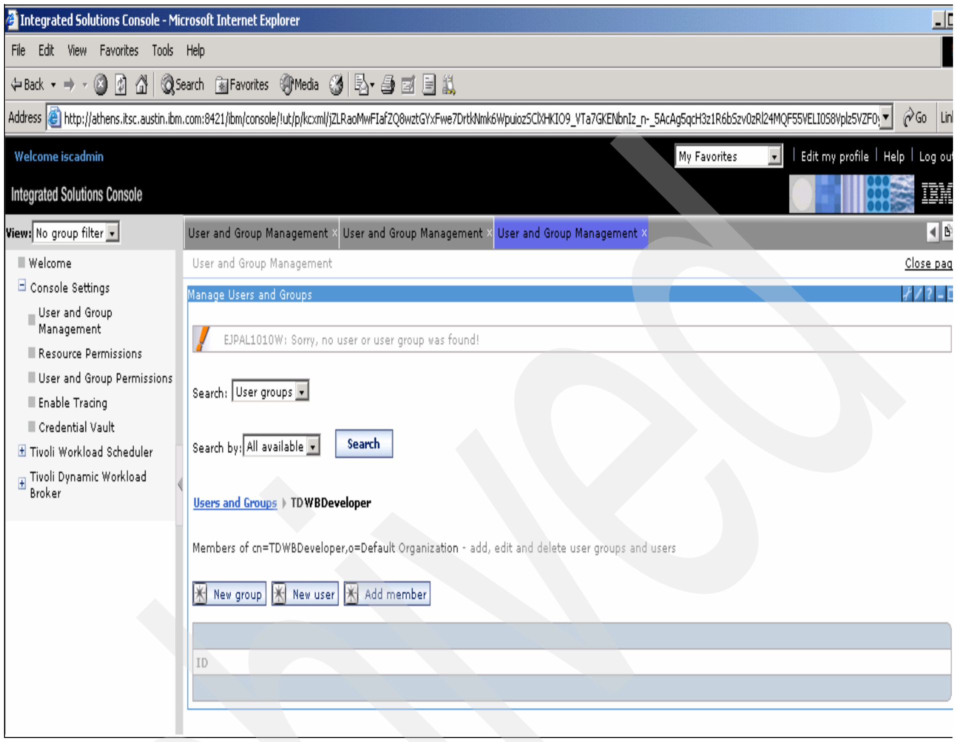

Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://athens.itsc.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/jZDfCoIwFIefxSc4x-amu1xGGSHZbJS7kREihn-6kKC3TyW9CcvtYh_n950DO6AhAV2bZ5GbtmhqU8IVNGtxOI5NPNthx8jfcbkKA4Fn7H2WY Go Links
Welcome iscadmin
Integrated Solutions Console
My Favorites | Edit my profile | Help | Log
View: No group filter
Welcome
Console Settings
User and Group Management
Resource Permissions
User and Group Permissions
Enable Tracing
Credential Vault
Tivoli Workload Scheduler
Tivoli Dynamic Workload Broker
User and Group Management
User and Group Management
User and Group Management
User and Group Management
User and Group Management
User and Group Management
Close page
Manage Users and Groups
Search: User groups
Search by: All available
Search
Users and Groups
New group
New user
Page 1 of 1
ID
All Portal User Groups
All Authenticated Portal Users
TDWBDeveloper
TWSWEBUIAdministrator
TDWBConfigurator
TWSWEBUIOperator
TDWBOperator
iscadmins
TDWBAdministrator
TWSWEBUIConfigurator
Page 1 of 1

Figure 5-24 ISC groups

In our example we add a user for TDWBDeveloper.

4. Click **ITDWBDevoper** and click **Add user**, as shown on Figure 5-25.


Integrated Solutions Console - Microsoft Internet Explorer

Address http://athens.itsc.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/jZLRaoMwFIafZQ8wztGYxFwe7DrtkNmk6Wpuioz5ClXHKIO9_VTa7GKENbnIz_n-_5AcAg5qcH3z1R6bS

Welcome iscadmin

Integrated Solutions Console

User and Group Management

Manage Users and Groups

EJPAL1010W: Sorry, no user or user group was found!

Search: User groups

Search by: All available

Users and Groups > TDWBDeveloper

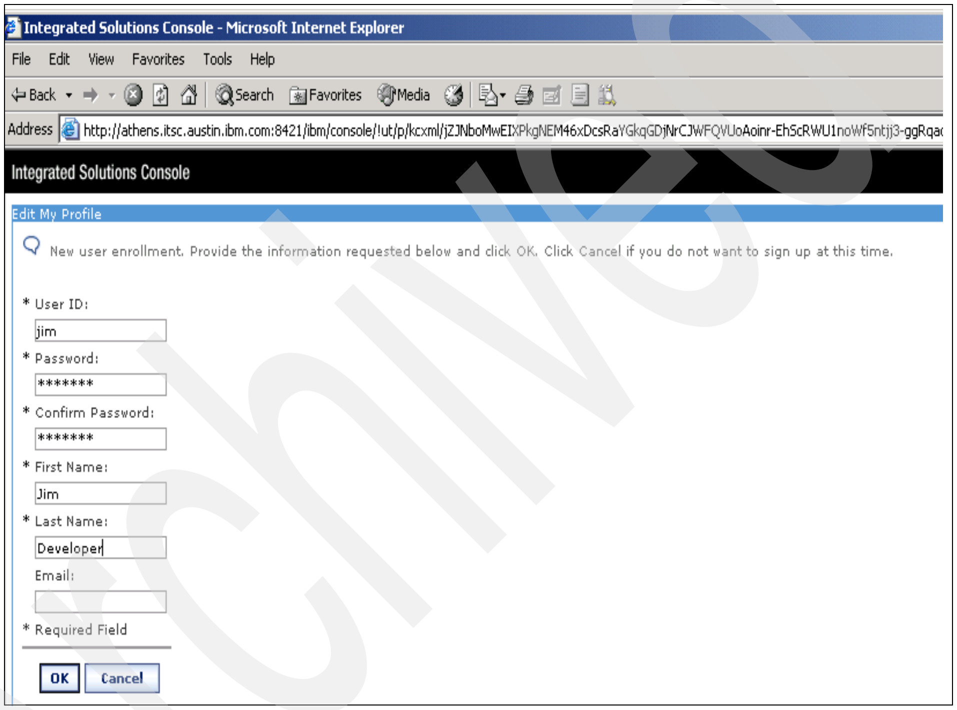
Members of cn=TDWBDeveloper,o=Default Organization - add, edit and delete user groups and users

File Edit View Favorites Tools Help
Back Search Favorites Media
Go
My Favorites | Edit my profile | Help | Log out
View: No group filter
Welcome
Console Settings
User and Group Management
Resource Permissions
User and Group Permissions
Enable Tracing
Credential Vault
Tivoli Workload Scheduler
Tivoli Dynamic Workload Broker
User and Group Management
Close page
Search
New group
New user
Add member
ID

Figure 5-25 Adding user for TDWBDeveloper

5. Fill in the information about the user, as shown on Figure 5-26. The user is now mapped to the Developer view.

> **Note:** The user name and password are not tied to any operating system or LDAP user. This is just the user name and password required for logging into the Tivoli Dynamic Workload Broker Web Console. Authorization for each user is determined by the user name and password defined on the Tivoli Dynamic Workload Broker on the server connection configuration.


Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://athens.itsc.austin.ibm.com:8421/ibm/console/!ut/p/kcxml/jZJNboMwEIXPkgNEM46xDcsRaYGkqGDjNrCJWFQVUoAoinr-EhScRWU1noWf5ntjj3-ggRqa
Integrated Solutions Console
Edit My Profile

New user enrollment. Provide the information requested below and click OK. Click Cancel if you do not want to sign up at this time.

* User ID: jim
* Password: *******
* Confirm Password: *******
* First Name: Jim
* Last Name: Developer
Email:
* Required Field

Figure 5-26 Adding a user to the TDWBDeveloper view

This concludes the addition of the user.

5.4 Command-line interface

The command-line interface allows authorized users to create, submit, and manage jobs on the Tivoli Dynamic Workload Broker Server.

For more information about user authentication and authorization for the command-line interface see 5.3.1, “Enabling global security” on page 220.

Installation details

CLI functionalities are invoked using command-line scipts. Scripts are installed in the directory *ITDWB_Server_install_directory*/bin during the server installation process.

CLI command property file

During the Tivoli Dynamic Workload Broker server installation process the installation module creates the *ITDWB_Server_Install_directory/config/*CLIConfig.properties file.

The CLIConfig.propoerties file contains configuration information that is used when typing commands. By default, arguments required when typing commands are retrieved from this file unless explicitly specified in the command syntax.

Information stored in the *ITDWB_Server_Install_directory/config/*CLIConfig.properties file is:

- ITDWBServerHost - specifies the IP address or host name of the Tivoli Dynamic Workload Broker server
- ITDWBServerPort - specifies the number of the Tivoli Dynamic Workload Broker server port
- Tivoli Dynamic Workload Broker server secureport
- use_secure_connection
- KeyStore file
- TrustStore file
- KeyStore password
- TrustStore password
- User ID to connect to ITDWB server webservices
- Password to connect to ITDWB server webservices
- CLI log level
- Database connect configuration

CLI functionalities

The CLI functionalities are:

- Manage job definitions → **`jobstore`**
- Job submission → **`jobsubmit`**
- Browse job output → **`jobgetexecutionlog`**
- Cancel job → **`jobcancel`**
- Getting single job details → **`jobdetails`**
- Query jobs → **`jobquery`**
- Archive database tables → **`movehistorydata`**

Environment configuration details

Before using the CLI function, the user must configure the environment. In order to perform this operation the user must launch the tdwb_env.bat (.sh) script.

5.4.1 jobstore

Invoking the jobstore.bat (sh) script allows an authorized usersto perform the following operations on job definitions:

- Save and update the job definition file in the JobRepository database.
- Delete job definitions.
- Print job definitions to standard output.
- Perform queries on job definitions.

The jobstore syntax is:

```
         jobstore [? | [-usr <username> -pwd <password>]
       { [ -create <jsdl_file> ] |
         [ -update <jsdl_file> ] |
         [ -del <job_definition_name> ] |
         [ -get <job_definition_name> ] |
         [ -queryname <job_definition_name> -querytarget
<job_definition_target>
-querydesc <job_definition_descr> -queryuser  <job_definition_user> ] }
         [ -configFile <configuration_file> ] ]
```

jobstore to save job definition file in the JobRepository

This command invokes the Tivoli Dynamic Workload Broker server Web services and saves the job definition file in the JobRepository database. It creates a single record in the jod_job_definition table, using as primary key value *<jobdefition name>* attribute value in job definition file. See the following example:

```
jobstore –create BA-WIN-JOB2.jsdl
```

Example 5-42 Using jobstore to add new job definition to JobRepository

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>edit BA-WIN-JOB2.jsdl

C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobstore -create BA-WIN-JOB2.jsdl
Call Job Dispatcher to save the job definition
Job Definition was successfully saved in Job Repository

C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobstore -queryname BA-WIN-JOB2
Call Job Dispatcher to query job definition
Success returned from Job Dispatcher
There are 1 Job Definitions found for your request.
Details are as follows:

Job Definition Name: BA-WIN-JOB2
Job Definition Description: Business Windows job version 2
Job Definition Owner: Administrator
Job Definition CreationTime: Wed May 09 10:33:47 CDT 2007
```

Jobstore to update job definition

This command invokes the Tivoli Dynamic Workload Broker server Web Services and updates the job definition previously saved in JobRepository database. See Example 5-43.

```
jobstore.bat –update jsdl_ping_win.xml
```

Example 5-43 Using jobstore to update existing job definition

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobstore -update BA-WIN-JOB2.jsdl
Call Job Dispatcher to set the job definition
Job Definition is successfully set in Job Repository
```

jobstore to delete job definition

This command invokes the Tivoli Dynamic Workload Broker server Web services, deleting the job definition in the JobRepository database. It invokes the Tivoli Dynamic Workload Broker server Web services, retreiving the job definition previously saved in JobRepository database. See Example 5-44.

```
jobstore –del BA-WIN-JOB2
```

Example 5-44 Using jobstore to delete an existing job definition

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobstore -del BA-WIN-JOB2
Call Job Dispatcher to delete the job definition
Job Definition BA-WIN-JOB2 was successfully deleted from Job Repository
```

jobstore to get job definition

This command invokes the Tivoli Dynamic Workload Broker server Web services, geting the job definition in the JobRepository database. See Example 5-45.

```
jobstore –get WeeklyReport
```

Example 5-45 jobstore -get command and output

```
C:\WINDOWS>jobstore -get WeeklyReport
Call Job Dispatcher to get job definition
Success returned from Job Dispatcher
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinitionType
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling
/1.0/jsdle" name="WeeklyReport">
  <jsdl:application name="executable">
    <jsdle:executable path="C:\WINDOWS\system32\whoami.exe"/>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:candidateHosts>
      <jsdl:hostName>Athenshjh</jsdl:hostName>
    </jsdl:candidateHosts>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="LINUX" version="4.3.2"/>
    </jsdl:candidateOperatingSystems>
  </jsdl:resources>
  <jsdl:scheduling>

<jsdl:maximumResourceWaitingTime>P0Y0M0DT0H0M30S</jsdl:maximumResourceW
aitingTime>
```

```
    <jsdl:priority>0</jsdl:priority>
  </jsdl:scheduling>
</jsdl:JobDefinitionType>
```

Jobstore to perform query on job definitions

This command invokes the Tivoli Dynamic Workload Broker server Web Services and performs a search on job definitions(jod_job_definitions table) based on job definition name, and returns information about the job.

Example 5-46 is an example:

```
jobstore –queryname BA*
```

Example 5-46 jobstore -queryname command and output

```
C:\WINDOWS>jobstore -queryname BA*
Call Job Dispatcher to query job definition
Success returned from Job Dispatcher
There are 4 Job Definitions found for your request.
Details are as follows:

Job Definition Name: BA-A-DB2QUERY1
Job Definition Description: Business area A  DB2 query job
Job Definition Owner: administrator
Job Definition CreationTime: Fri Apr 13 18:35:22 CDT 2007
Job Definition ModificationTime: Fri Apr 13 18:47:54 CDT 2007

Job Definition Name: BA-A-Linuxjob
Job Definition Description: Business area A Linux job
Job Definition Owner: administrator
Job Definition CreationTime: Fri Apr 13 18:57:19 CDT 2007
Job Definition ModificationTime: Fri Apr 13 18:57:19 CDT 2007


Job Definition Name: BA-B-DB2QUERY1
Job Definition Description: Business area B  DB2 query job
Job Definition Owner: jim
Job Definition CreationTime: Fri Apr 13 18:43:21 CDT 2007
Job Definition ModificationTime: Thu Apr 26 20:22:21 CDT 2007
Job Definition Name: BA-C-Windows-job
Job Definition Description: Business area C Windows job
Job Definition Owner: administrator
Job Definition CreationTime: Fri Apr 13 19:01:06 CDT 2007
Job Definition ModificationTime: Fri Apr 13 19:01:06 CDT 2007
```

5.4.2 jobsubmit

This command submits jobs using the jobsubmit script. Invoking the jobsubmit.bat (sh) script allows an authorized user to submit a job to the job dispatcher.

The **`jobsubmit`** syntax is:

```
jobsubmit [? | [-usr <username> -pwd <password>]
        {-jsdl <jsdl_file> | -jdname <job_definition_name>}
         [-alias <job_alias>]
      [-var <variable=value>]...
        [-affinity {jobid=<job_id>|alias=<job_alias>}]
       [-configFile <configuration_file>] ]
```

Submitting .jsdl job definitions

For example:

```
jobsubmit –jsdl WinJob.jsdl
```

- This command invokes the Tivoli Dynamic Workload Broker server Web services and submits a .jsdl job definition (not stored in the JobRepository database) to the server.
- A unique ID is assigned to the job.
- Job status and details are saved in job_jobs table.
- The job is now ready to be processed by Tivoli Dynamic Workload Broker server.
- The job definition is *not* added to the JobRepository.

Example 5-47 jobstore command for submitted job not stored in JobRepository

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobsubmit -jsdl WinJob.jsdl
Call Job Dispatcher to submit the job.
Success returned from Job Dispatcher
The job f3f6facc-d0e5-3e8e-a1c1-32baed7d3472 submitted successfully

C:\DOCUME~1\ADMINI~1\JD_WOR~1>
```

Submitting jobs stored in the JobRepository

The command invokes the Tivoli Dynamic Workload Broker server Web services and submits a job using its job definition name previously saved in a database to the server. For example:

```
jobsubmit.bat –jdname BA-C-Windows-job
```

Example 5-48 jobsubmit command

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobsubmit -jdname BA-C-Windows-job
Call Job Dispatcher to submit the job.
Success returned from Job Dispatcher
The job bde3b751-f11a-3c54-85f0-7fd73d55b6e9 submitted successfully

C:\DOCUME~1\ADMINI~1\JD_WOR~1>
```

5.4.3 jobgetexecutionlog

You can obtain job output for submitted jobs using the jobgetexecutionlog script.

Invoking the jobgetexecutionlog.bat (sh) script allows an authorized user to obtain job output on a submitted job using the unique ID created at job submission.

The **jobgetexecutionlog** syntax is:

```
        jobgetexecutionlog [? | [-usr <username> -pwd <password>]
                {-id <job_id> -sizePage <Page Size> -offset <offset>}
              [-configFile <configuration_file>] ]
```

Using jobgetexecutionlog command

This command invokes the Tivoli Dynamic Workload Broker server Web services, retrieving job output for jobs submitted to the server. For example:

```
jobgetexecutionlog –id 2296ad5e-beb7-342c-ae27-1238a83fe0b8 -sizePage
400 -offset 1
```

Example 5-49 Retrieving job output with jobgetexecutionlog command

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobsubmit -jdname linuxDir
Call Job Dispatcher to submit the job.
Success returned from Job Dispatcher
The job 2296ad5e-beb7-342c-ae27-1238a83fe0b8 submitted successfully

C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobgetexecutionlog -id
2296ad5e-beb7-342c-ae27-1238a83fe0b8 -sizePage 400 -offset 1
```

```
Call Job Dispatcher to get the output of the job
Success returned from Job Dispatcher
Get Execution Log request submitted
The Execution Log Page requested is:
 otal 0
drwxrwxrwx  6 root root 248 Apr 26 06:38 eclipse
drwxr-xr-x  2 root root 216 Apr 26 06:40 logs
drwxrwxrwx  5 root root 240 Apr 26 06:37 rcp
drwxrwxrwx  3 root root  72 Apr 26 06:37 shared
drwxr-xr-x  3 root root  80 Apr 26 06:38 workspace

The file size is 247 bytes.
```

5.4.4 jobcancel

Invoking the jobcancel.bat (sh) script allows the user to cancel an executing job previously submitted to the server. The job will be canceled only if it is in one of the following states:

- SUBMITTED
- WAITING_FOR_RESOURCES
- RESOURCE_ALLOCATION_RECEIVED
- SUBMITTED_TO_ENDPOINT
- RESOURCE_REALLOCATE
- EXECUTING

The **jobcancel** syntax is:

```
jobCancel [? | [-usr <username> -pwd <password>]
      -id <job_id>
      [-configFile <configuration File>]]
```

Using jobcancel script

The command cancels the running of submitted jobs using the unique ID created at job submission. To retrieve the job ID after submitting the job, you can use the **jobquery** command specifying the job name. For example:

```
jobcancel –id 9e4465cb-cfca-3077-8ba1-178c3a4ea16d
```

Example 5-50   Cancelling newly submitted job using jobcancel

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobsubmit -jdname SimpleSleep2
Call Job Dispatcher to submit the job.
Success returned from Job Dispatcher
The job 9e4465cb-cfca-3077-8ba1-178c3a4ea16d submitted successfully
```

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobquery -name SimpleSleep2
Call Job Dispatcher to query jobs
Success returned from Job Dispatcher

There are 1 Jobs found for your request
Details are as follows:

Job Name: SimpleSleep2
Job Alias: N/A
Job ID: 9e4465cb-cfca-3077-8ba1-178c3a4ea16d
Job Status: EXECUTING
Job EPR: http://athens:9550/JDServiceWS/services/Job
Job Submitter: Administrator
Job Submitter Type: N/A
Job Submit Time: Wed May 09 13:48:41 CDT 2007
Job Start Time: Wed May 09 12:37:37 CDT 2007
Job Last Status Message: N/A
Job Duration: N/A
Job Return Code: 0
Job Resource Name: barcelona.itsc.austin.ibm.com
Job Resource Type: ComputerSystem


C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobcancel -id
9e4465cb-cfca-3077-8ba1-178c3a4ea16d
Call Job Dispatcher to cancel the job
Success returned from Job Dispatcher
Job cancel request submitted


C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobquery -name SimpleSleep2
Call Job Dispatcher to query jobs
Success returned from Job Dispatcher

There are 1 Jobs found for your request
Details are as follows:

Job Name: SimpleSleep2
Job Alias: N/A
Job ID: 9e4465cb-cfca-3077-8ba1-178c3a4ea16d
Job Status: CANCELLED
Job EPR: http://athens:9550/JDServiceWS/services/Job
Job Submitter: Administrator
Job Submitter Type: N/A
Job Submit Time: Wed May 09 13:48:41 CDT 2007
Job Start Time: Wed May 09 12:37:37 CDT 2007
Job End Time: Wed May 09 12:38:56 CDT 2007
Job Last Status Message: N/A
Job Duration: 0d 0h 1m 19s 0ms
Job Return Code: 0
Job Resource Name: barcelona.itsc.austin.ibm.com
Job Resource Type: ComputerSystem
```

5.4.5 jobdetails

Invoking the jobdetails.bat (sh) script allows an authorized user to view details on a submitted job using the unique ID created at job submission.

The **`jobdetails`** syntax is:

```
jobDetails [? | [-usr <username> -pwd <password>]
      -id <job_id>
      [-v]
      [-configFile <configuration_File>] ]
```

Using jobdetails script

This command invokes the Tivoli Dynamic Workload Broker server Web services, showing the job details information. Passing the –v parameter script provides more detailed information. For example:

```
jobdetails –id 5a116c18-af44-33c0-a37f-afc56d72d922 -v
```

Example 5-51 Obtaining summary and detail job information using the jobdetails command

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobsubmit -jdname SimpleSleep
Call Job Dispatcher to submit the job.
Success returned from Job Dispatcher
The job 5a116c18-af44-33c0-a37f-afc56d72d922 submitted successfully

C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobdetails -id
5a116c18-af44-33c0-a37f-afc56d72d922
Call Job Dispatcher to get the job properties
Success returned from Job Dispatcher

Job ID: 5a116c18-af44-33c0-a37f-afc56d72d922
Job Status: EXECUTING

C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobdetails -id
5a116c18-af44-33c0-a37f-afc56d72d922 -v
```

```
Call Job Dispatcher to get the job properties
Success returned from Job Dispatcher

Job ID: 5a116c18-af44-33c0-a37f-afc56d72d922
Job Name: SimpleSleep
Job Alias: N/A
Job Status: EXECUTING
Job Submitter: Administrator
Job Submitter Type: TDWB CLI
Client notification: N/A
Job Last Status Message: N/A
Job Submit Time: Wed May 09 13:57:06 CDT 2007
Job Start Time: Wed May 09 05:27:15 CDT 2007
Job End Time: N/A
Job Duration: N/A
Job Return Code: N/A
Job Resource Name: oslo
Job Resource Type: ComputerSystem
```

5.4.6 jobquery

Invoking the jobquery.bat (or sh) script allows an authorized user to perform an advanced query on a submitted job based on the following attributes:

- Job status:
  - 0 - all supported job states
  - 1 - all submitted
  - 2 - waiting for resources
  - 3 - resource allocation received
  - 4 - submitted to agent
  - 5 - running
  - 6 - cancel pending
  - 7 - canceling reallocation
  - 41 - resource allocation failed
  - 42 - run failed
  - 43 - competed successfully
  - 44 - canceled
  - 45 - unknown job
  - 46 - job not started
- Name of the user who submitted the job
- Job name
- Job completion date

The **jobquery** syntax is:

```
jobquery [? | [-usr <username> -pwd <password>]
 { [ -status <status number> |]
   [ -submitter <submitter>]
   [ -name <job_definition_name> ]
   [ -alias <job_alias> ]
   [ -sbf <submit_date_from> ]
   [ -sbt <submit_date_to> ]
   [ -jsdf <job_start_date_from> ]
   [ -jsdt <job_start_date_to> ]
   [ -jedf <job_start_date_from> ]]
   [ -jedt <job_start_date_to> ]] }
   [ -configFile <configuration_file> ] ]
```

Using the jobquery script

For example:

```
jobquery.bat –status 44
```

This command invokes the Tivoli Dynamic Workload Broker server Web services and shows details for all jobs in a CANCELLED state in the job dispatcher database. See Example 5-52.

Example 5-52 Using the jobquery command

```
C:\DOCUME~1\ADMINI~1\JD_WOR~1>jobquery -status 44
Call Job Dispatcher to query jobs
Success returned from Job Dispatcher

There are 2 Jobs found for your request
Details are as follows:

Job Name: SimpleSleep
Job Alias: N/A
Job ID: 3923e506-78ed-3ad4-ae39-92903cf26146
Job Status: CANCELLED
Job EPR: http://athens:9550/JDServiceWS/services/Job
Job Submitter: Administrator
Job Submitter Type: N/A
Job Submit Time: Wed May 09 11:21:02 CDT 2007
Job Start Time: Wed May 09 02:51:22 CDT 2007
Job End Time: Wed May 09 02:52:15 CDT 2007
Job Last Status Message: N/A
Job Duration: 0d 0h 0m 53s 0ms
Job Return Code: 0
Job Resource Name: oslo
Job Resource Type: ComputerSystem

Job Name: SimpleSleep2
Job Alias: N/A
Job ID: 9e4465cb-cfca-3077-8ba1-178c3a4ea16d
Job Status: CANCELLED
Job EPR: http://athens:9550/JDServiceWS/services/Job
Job Submitter: Administrator
Job Submitter Type: N/A
Job Submit Time: Wed May 09 13:48:41 CDT 2007
Job Start Time: Wed May 09 12:37:37 CDT 2007
Job End Time: Wed May 09 12:38:56 CDT 2007
Job Last Status Message: N/A
Job Duration: 0d 0h 1m 19s 0ms
Job Return Code: 0
Job Resource Name: barcelona.itsc.austin.ibm.com
Job Resource Type: ComputerSystem
```

5.4.7 movehistorydata

Archive a database table using the movehistorydata script. A user can use the movehistorydata script to move the data from the JobRepository database to the archive database tables. Job are moved in the following database tables:

- JOA_JOBS_ARCHIVES
- JRA_JOB_RESOURCE_ARCHIVES
- MEA_METRIC_ARCHIVES

The **movehistorydata** syntax is:

```
   MoveHistoryData [? | [-dbUsr <username> -dbPwd <password>]
-successfulJobsMaxAge <successfulJobsMaxAge>
             -notSuccessfulJobsMaxAge <notSuccessfulJobsMaxAge>
-archivedJobsMaxAge <- -archivedJobsMaxAge > ]
```

Using movhistorydata script

The following is an example of movhistorydata script usage:

```
movehistorydata.bat -successfulJobsMaxAge 1 - notSuccessfulJobsMaxAge 2
- archivedJobsMaxAge 4
```

Example 5-53 movhistorydata script

```
1\JD_WOR~1>movehistorydata -successfulJobsMaxAge 0
-notSuccessfulJobsMaxAge 0 -archivedJobsMaxAge 0 -dbUsr db2admin -dbPwd
itso05

History data successfully moved
1\JD_WOR~1>
```

6

High availability and recovery considerations

In this chapter we look at the configuration of the Tivoli Dynamic Workload Broker server in a high availability cluster that includes DB2, WebSphere Application server, and Tivoli Dynamic Workload Broker server. This chapter focuses on the installation and configuration of all of the products that are prerequisites for the Tivoli Dynamic Workload Broker server using Tivoli System Automation. We cover the following topics in detail:

6.1 High-availability scenario

The only scenario that we describe in this chapter is passive-active failover, where there is a single Tivoli System Automation cluster with one active master node and one standby node. All three applications (namely DB2, WebSphere Application server, and Tivoli Dynamic Workload Broker server) are installed on both the master and standby node local disks. Tivoli System Automation monitors these three applications, and if there is a problem then Tivoli System Automation moves the application from one side to the other side by stopping the application and then starting it on the other side.

A passive-active high availability scenario is shown in 6.1, “High-availability scenario” on page 248.

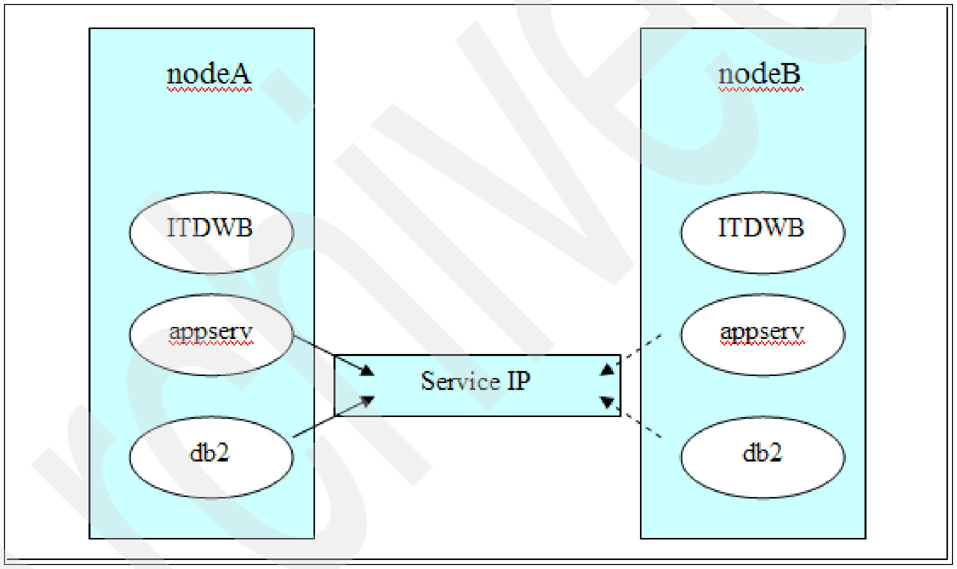


Figure 6-1 Passive-active HA scenario

6.2 IBM Tivoli System Automation for Multiplatforms

IBM Tivoli System Automation for Multiplatforms (Tivoli System Automation) is a product that provides high availability (HA) by automating the control of IT resources such as processes, file systems, IP addresses, and other resources. It facilitates the automatic switching of users, applications, and data from one system to another in the cluster after a hardware or software failure.

6.2.1 How Tivoli System Automation works

The Tivoli System Automation product provides HA by automating resources such as processes, applications, and IP addresses. To automate an IT resource (for example, an IP address), the resource must be defined to Tivoli System Automation. Every application must be defined as a resource in order to be managed and automated with Tivoli System Automation. Application resources are usually defined in the generic resource class IBM.Application. For the HA IP address, the resource class IBM.ServiceIP must be used.

Reliable Scalable Cluster Technology (RSCT) is a product fully integrated into Tivoli System Automation. RSCT is a set of software products that together provide a comprehensive clustering environment for AIX and Linux. RSCT is the infrastructure to provide clusters with improved system availability, scalability, and ease of use. RSCT provides three basic components, or layers, of functionality:

- Resource Monitoring and Control (RMC) provides global access for configuring, monitoring, and controlling resources in a peer domain.
- High Availability Group Services (HAGS) is a distributed coordination, messaging, and synchronization service.
- High Availability Topology Services (HATS) provides a scalable heartbeat for adapter and node failure detection, and a reliable messaging service in a peer domain.

Terminology

Some of the key terms used in describing Tivoli System Automation are:

- Cluster or peer domain

  The group of host systems upon which Tivoli System Automation manages resources is known as a cluster. A cluster can consist of one or more systems or nodes.

- Resource

  A resource is any piece of hardware or software that can be defined to Tivoli System Automation. These resources can be either defined manually by the administrator using the **`mkrsrc`** (make resource) command or through the *harvesting* functionality of the cluster infrastructure, whereby resources are automatically detected and prepared for use. All resources are controlled through the appropriate resource managers.

- Resource class

  A resource class is a collection of resources of the same type. For example, if an application is a resource, then all applications defined in the cluster would comprise a resource class. Resource classes allow you to define the common characteristics among the resources in its class. In the case of applications, the resource class can define identifying characteristics, such as the name of the application, and varying characteristics, such as whether the application is running. So each resource in the class can then be noted by its characteristics at any given time.

- Resource group

  Resource groups are logical containers for a collection of resources. This container allows you to control multiple resources as a single logical entity. Resource groups are the primary mechanism for operations within Tivoli System Automation. Resource groups can also be nested, meaning that applications can be split into several resource groups, which themselves are part of another higher-level resource group. Also, resource groups can be defined in such a way that their members can be located on different systems in the cluster.

- Managed resource

  A managed resource is a resource that has been defined to Tivoli System Automation. To accomplish this, the resource is added to a resource group, at which time it becomes manageable through Tivoli System Automation.

- Nominal state

  The nominal state of a resource group indicates to Tivoli System Automation whether the resources with the group should be online or offline at this point in time. So setting the nominal state to offline indicates that you wish for Tivoli System Automation to stop the resources in the group, and setting the nominal state to online.is an indication that you wish to start the resources in the resource group.

- Equivalency

  An equivalency is a collection of resources that provide the same functionality. For example, equivalencies are used for selecting network adapters that should host an IP address. If one network adapter goes offline, Tivoli System Automation selects another network adapter to host the IP address.

- Relationships

  Tivoli System Automation allows for the definition of relationships between resources in a cluster. There are two different relationship types:

  – Start/stop relationships

    These relationships are used to define start and stop dependencies between resources. You can use the StartAfter, StopAfter, DependsOn, DependsOnAny, and ForcedDownBy relationships to achieve this. For example, a resource must only be started after another resource was started. You can define this by using the policy element StartAfter relationship.

- Location relationships

  Location relationships are applied when resources must, or should if possible, be started on the same or a different node in the cluster.

- Resource manager

  Resource classes are managed by the various resource managers (RMs), depending on what type of resource is being managed. A resource manager is a software layer between a resource and RMC. The following resource managers are provided by Tivoli System Automation:

  - Recovery RM (IBM.RecoveryRM)

    This resource manager serves as the decision engine for Tivoli System Automation. When a policy for defining resource availability and relationships is defined, this information is supplied to the Recovery RM. This RM runs on every node in the cluster, with exactly one Recovery RM designated as the master. The master evaluates the monitoring information from the various resource managers. When a situation develops that requires intervention, the Recovery RM drives the decisions that result in start or stop operations on the resources as needed.

  - Global Resource RM

    The Global Resource RM (IBM.GblResRM) supports two resource classes:

    - IBM.Application

      The IBM.Application resource class defines the behavior for general application resources. This class can be used to start, stop, and monitor processes. As a generic class, it is very flexible and can be used to monitor and control various kinds of resources. Most of the applications that you automate are done using this class.

    - IBM.ServiceIP

      This application class defines the behavior of Internet Protocol (IP) address resources. It allows you to assign IP addresses to an adapter. In effect, it allows IP addresses to *float* among nodes.

  - Configuration RM

    The Configuration RM (IBM.ConfigRM) is used in the cluster definition. In addition, quorum support, which is a means of insuring data integrity when portions of a cluster lose communication, is provided.

  - Event Response RM

    The Event Response RM (IBM.ERRM) provides the ability to monitor conditions in the cluster in order for the RMC system to react in certain ways.

– Test RM

  The Test resource manager (IBM.TestRM) manages test resources and provides functions to manipulate the operational state of these resources. The resource manager is operational in a peer domain mode only and provides the resource class IBM.Test.

For more detailed information about Tivoli System Automation Resource Managers see *IBM Tivoli System Automation for Multiplatforms Base Component User's Guide*, SC33-8210.

6.2.2 Installing and configuring Tivoli System Automation

In this section we look at installing and configuring Tivoli System Automation. For more detailed information refer to *IBM Tivoli System Automation for Multiplatforms Base Component User's Guide*, SC33-8210.

Before installing Tivoli System Automation make sure that both nodes are able to communicate with each other using host names and fixed IP addresses. Insert both names in the /etc/hosts file on both systems. In this manner, Domain Name System (DNS) server failures outside of the cluster do not affect the high availability of the cluster.

For our discussion, we use the nodes edinburgh (as nodeA) and prague (as nodeB). The IP address of edinburough is 9.3.5.169 and the IP address of prague is 9.3.5.97. We also set up three alias IP addresses for the three products (namely, DB2, WebSphere Application Server, and Inter-grated Solutions console, which are using 9.3.5.90, 9.3.5.91, and 9.3.5.93 respectfully). The hosts filed on both sides of the cluster looked like Example 6-1.

Example 6-1 The /etc/hosts file

```
127.0.0.1                localhost.localdomain localhost
9.3.5.169                edinburgh.itsc.austin.ibm.com edinburgh nodeA
9.3.5.97                 prague.itsc.austin.ibm.com prague nodeB
9.3.5.90                 db2
9.3.5.91                 was
9.3.5.93                 isc
```

Installation of Tivoli System Automation

Installation steps are as follows:

1. If you downloaded the tar file from the Internet, extract the file using the following command:

   ```
   tar -xvf <tar file>
   ```

If you got the product on a CD, mount the CD and change to the directory where the CD is mounted. Now change to the appropriate directory.

2. Install the product on each system in the cluster with the installSAM script as the root user:

   ```
   # ./installSAM
   ```

3. Tivoli System Automation requires that a valid product license is installed on each system it is running on. The license is contained on the installation medium in the license sub directory. The installation of the license is usually performed during the product installation process. In case this did not succeed, or you want to upgrade from a Try & Buy license to a full license of the product, issue the following command to install the license:

   ```
   # samlicm –i license_file
   ```

   In order to display the license, issue:

   ```
   # samlicm -s
   ```

4. After installation, ensure that the variable CT_MANAGMENT_SCOPE is always set to 2 in the shell. Consider inserting this into the system-wide shell profile.
5. Install the fix packs in a similar manner on both systems in the cluster, as shown in Example 6-2.

   Example 6-2 Upgrading Tivoli System Automation

   ```
   [root@edinburgh]# CT_MANAGEMENT_SCOPE=2
   [root@edinburgh]# export CT_MANAGEMENT_SCOPE
   [root@edinburgh]# cd /root/code/SA/SAM2111Base
   [root@edinburgh]# ./installSAM
   ```

6. Install the latest Tivoli System Automation policies, as shown in Example 6-3. You can download them from the following Web site:

   ftp://ftp.software.ibm.com/software/tivoli/products/sys-auto-linux/

   Example 6-3 Installing Tivoli System Automation policies

   ```
   [root@edinburgh]# rpm -ivh sam.policies-1.2.2.1-06212.i386.rpm
   ```

Configuration of Tivoli System Automation

Configuring Tivoli System Automation to automate or to manage resources involves the following basic steps:

1. Prepare both nodes.

   preprpnode - This command prepares the security settings for the node to be included in a cluster. When issued, public keys are exchanged among the nodes, and the RMC access control list (ACL) is modified to enable access to cluster resources by all of the nodes of the cluster.

2. Create a Tivoli System Automation Domain.

   `mkrpdomain` - This command creates a new cluster definition. It is used to specify the name of the cluster and the list of nodes to be added to the cluster.

3. Create a resource group.

   `reghadrsalin` - This command can be used to create a resource group.

4. Add resources to the resource group.

   `addrgmbr` - This command adds one or more resources to a resource group.

5. Create equivalencies (typically used for IP address resources).

   `mkequ` - This command makes an equivalency resource.

6. Specify dependencies.

   `mkrel` - This command is used to make dependencies.

In order to create a new resource of the application resource class (for example, for the Deployment Manager), the following three scripts (or commands, respectively) must be provided:

1. A start script (or command) to bring the resource online
2. A stop script (or command) to take the resource offline
3. A script (or command) to monitor the resource through polling

To configure Tivoli System Automation:

1. Prepare both of the nodes and create the cluster by running the commands shown in Example 6-4.

   Example 6-4 Preparing the Tivoli System Automation domain

   ```
   [root@edinburgh]# preprpnode edinburgh prague
   [root@edinburgh]# mkrpdomain itsoDomain edinburgh prague
   [root@edinburgh]# startrpdomain itsoDomain
   ```

2. Wait for the domain to come online. To check that the domain has started and is online you can use the command **`lsrpdomain`**, as shown in Example 6-5.

   Example 6-5 Checking that the domain is online

   ```
   [root@edinburgh]# /usr/sbin/rsct/bin/lsrpdomain
   Name       OpState RSCTActiveVersion MixedVersions TSPort GSPort
   itsoDomain Online  2.4.6.2           No            12347  12348
   ```

3. Define a tie breaker. Tie breakers are necessary on clusters where there is an even number of nodes. This is to ensure that the cluster remains operational

even when there is a total communication failure or node failure. The different types of tie breakers and how to configure them are explained in Chapter 10, “Protecting your resources - quorum support,” of the *IBM Tivoli System Automation for Multiplatforms Base Component User's Guide*, SC33-8210. Use a network tie breaker that points to the gateway used by both cluster nodes, as shown in Example 6-6. Note that the keywords are case-sensitive.

Example 6-6 Define a tie breaker

```
[root@edinburgh]# mkrsrc IBM.TieBreaker Type=“EXEC” Name=“netTieBrk” \
DeviceInfo='PATHNAME=/usr/sbin/rsct/bin/samtb_net \
Address=9.3.5.255 Log=1' PostReserveWaitTime=30
```

4. Activate the tie breaker with the following command:

```
[root@edinburgh]# chrsrc -c IBM.PeerNode \
OpQuorumTieBreaker=“netTieBrk”
```

5. Define a netmon.cf file on each node. Create a file named netmon.cf in the /usr/sbin/cluster directory. In this file, place the IP address of each network interface's gateway, one address per line, as shown in Example 6-7.

Example 6-7 The netmon.cf file

```
9.3.5.255
```

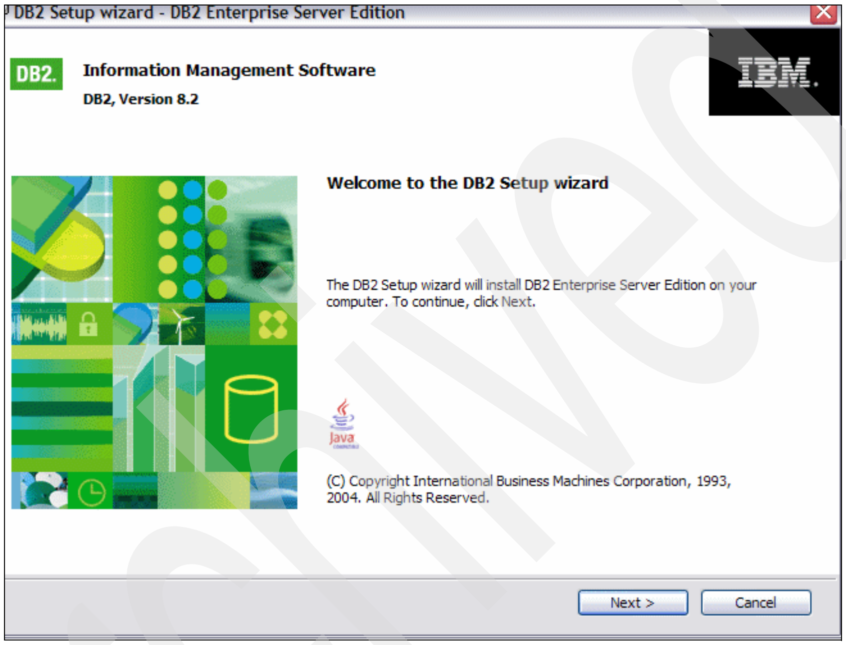
6.3 Installing and configuring DB2

The first step is to install DB2 Universal Database on both nodes. In this example we use the graphical installation for DB2, and for the most part use default selections.

6.3.1 Installation DB2 UDB

To install DB2 Universal Database on both nodes, perform the following steps:

1. Log in as root, and from the installation media run **db2setup** to install DB2. Perform this step on both sides of the cluster using the same configuration. The wizard runs and you will be presented with the DB2 setup welcome screen, as shown in Figure 6-2.


DB2 Setup wizard - DB2 Enterprise Server Edition
DB2.
Information Management Software
DB2, Version 8.2
IBM.
Java
Next >
Cancel

Welcome to the DB2 Setup wizard

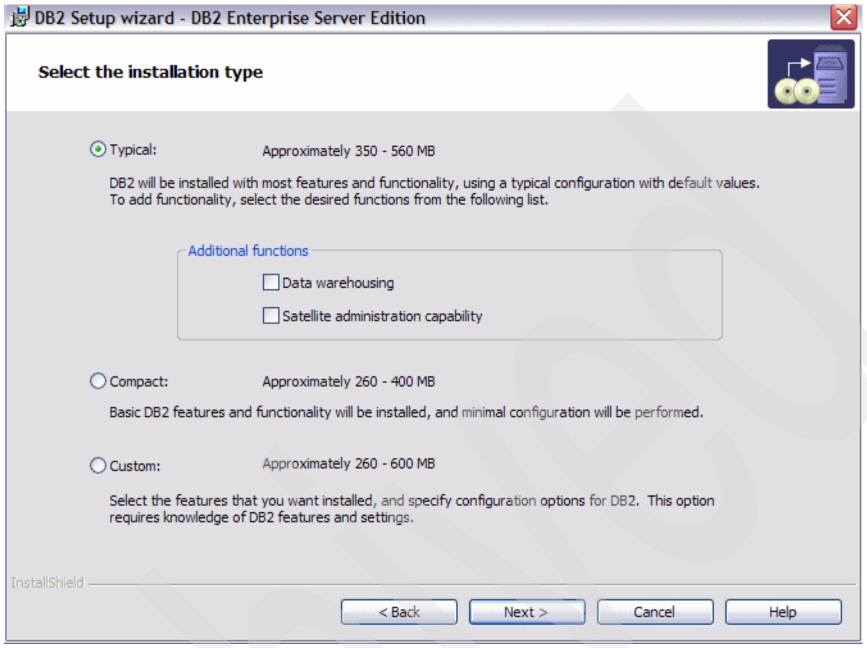
The DB2 Setup wizard will install DB2 Enterprise Server Edition on your computer. To continue, click Next.

(C) Copyright International Business Machines Corporation, 1993, 2004. All Rights Reserved.

Figure 6-2 DB2 welcome screen

2. On the next screen accept the license agreement and click **Next**.

3. On the next screen select the typical install, as shown in Figure 6-3, then click **Next**.



Select the installation type

[✓ Typical:] Approximately 350 - 560 MB

DB2 will be installed with most features and functionality, using a typical configuration with default values. To add functionality, select the desired functions from the following list.

Additional functions

☐ Data warehousing

☐ Satellite administration capability

○ Compact: Approximately 260 - 400 MB

Basic DB2 features and functionality will be installed, and minimal configuration will be performed.

○ Custom: Approximately 260 - 600 MB

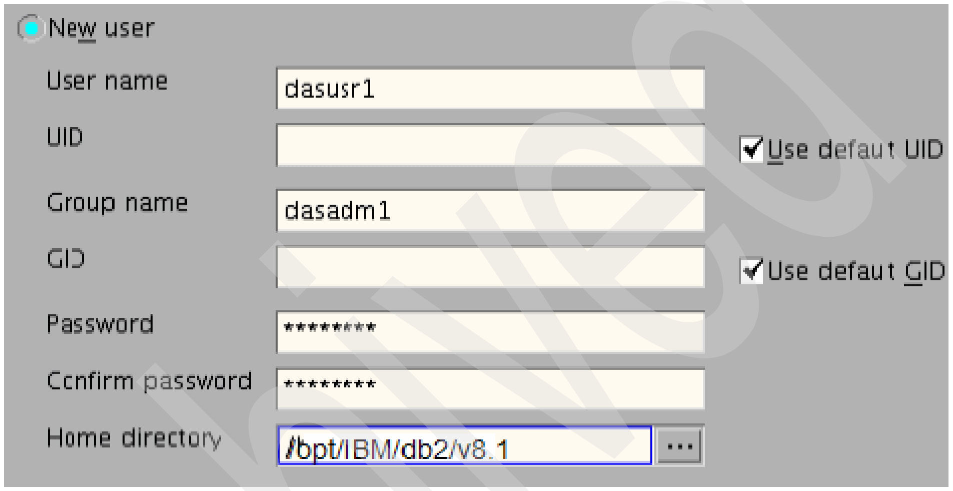
Select the features that you want installed, and specify configuration options for DB2. This option requires knowledge of DB2 features and settings.

Figure 6-3 DB2 install; Select the type of installation

4. On the next screen check the **Install DB2 Enterprise Server Edition** then click **Next**.

5. On the next screen you are presented with the user information. Use the same DB2 Administration Server (new user) (dasusr1, db2iadm1) with /opt/IBM/db2/v8.1 as the same home directories supplied, as shown in Figure 6-4. Then click **Next**.

Tip: Keep the installation the same on both sides of the cluster. Also remember the supplied password.



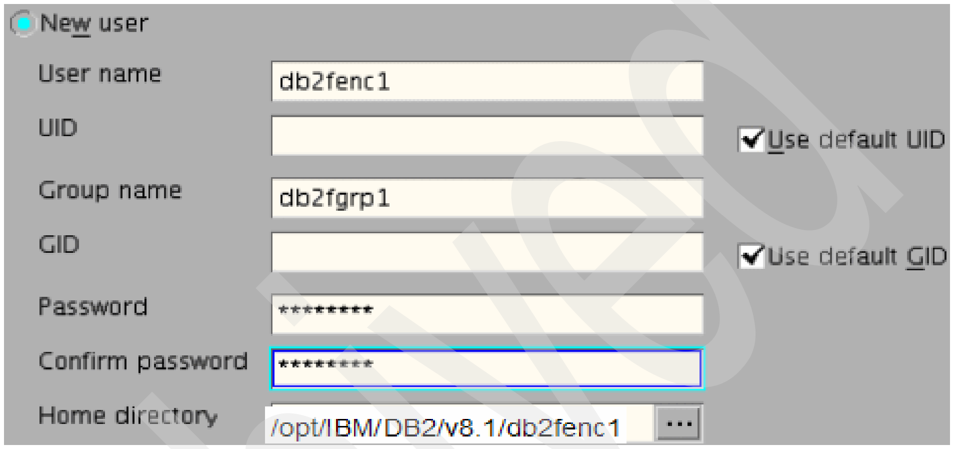
New user

User name	dasusr1	
UID		✓ Use defaut UID
Group name	dasadm1	
GID		✓ Use defaut GID
Password	*********	
Confirm password	*********	
Home directory	/opt/IBM/db2/v8.1	

Figure 6-4 Creating the DB2 Administration Server user

6. Create the fenced user (db2fenc1) and fenced group (db2fadm1) during this process, with the directory /opt/IBM/DB2/v8.1/db2fenc1, as shown in Figure 6-5.

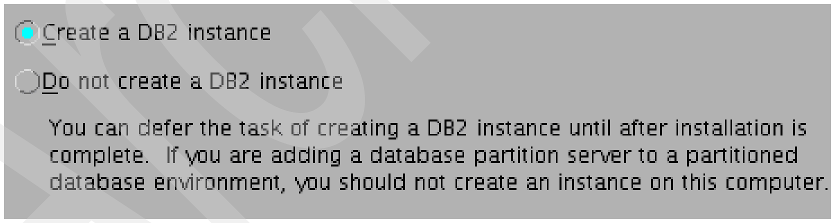
Tip: Keep the installation the same on both sides of the cluster. Also remember the supplied password.


[● New user]

User name	db2fenc1	
UID		✓ Use default UID
Group name	db2fgrp1	
GID		✓ Use default GID
Password	*********	
Confirm password	*********	
Home directory	/opt/IBM/DB2/v8.1/db2fenc1	

Figure 6-5 Creating the DB2 fence user

7. You will be prompted to create a DB2 instance. Select **Create a DB2 instance**, as shown in Figure 6-6. Then click **Next**.


[● Create a DB2 instance]

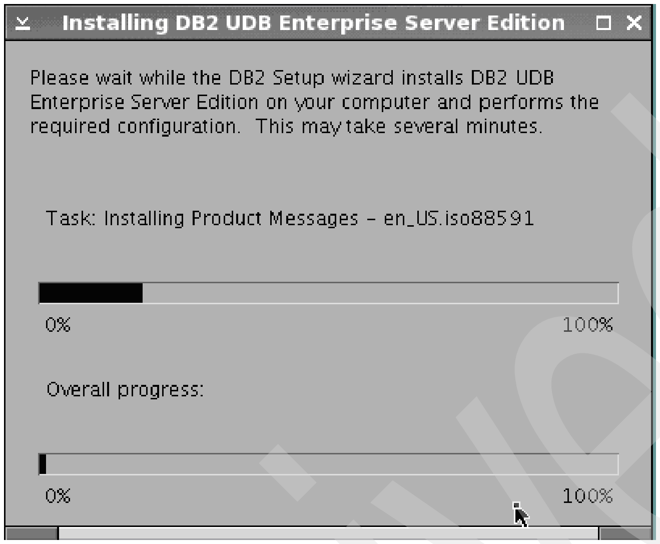
○ Do not create a DB2 instance

You can defer the task of creating a DB2 instance until after installation is complete. If you are adding a database partition server to a partitioned database environment, you should not create an instance on this computer.

Figure 6-6 Creating a DB2 instance

8. On the next screen select **Single-partition instance**, then click **Next**.

You are then presented with a progress installation bar, as shown in Figure 6-7.



Please wait while the DB2 Setup wizard installs DB2 UDB Enterprise Server Edition on your computer and performs the required configuration. This may take several minutes.

Task: Installing Product Messages - en_US.iso88591

Figure 6-7 DB2 installation progress bar

9. The install wizard then presents a setup complete screen. Check the **Status report** tab on the final setup panel to ensure that your installation is complete and correct, then click **Finish**.

6.3.2 Configuration of DB2 for Tivoli System Automation

In this section we go through the steps to configure DB2 for Tivoli System Automation so that Tivoli System Automation monitors and manages the DB2 failover. After this is in place, do not manually issue DB2 takeover commands. Instead, use the appropriate Tivoli System Automation commands to perform this function.

1. First ensure that hadr_domain is online by issuing the following command, as shown in Example 6-8.

```
[root@edinburgh ~]# lsrpdomain
```

Example 6-8 List of all Tivoli System Automation domains

```
[root@edinburgh ~]# lsrpdomain
Name       OpState RSCTActiveVersion MixedVersions TSPort GSPort
itsoDomain Online 2.4.6.2           No            12347 12348
```

2. Ensure that all nodes are online in the domain, as shown in Example 6-9.

Example 6-9 List of all Tivoli System Automation nodes

```
[root@edinburgh ~]# lsrpnode
Name      OpState RSCTVersion
prague    Online  2.4.6.2
edinburgh Online  2.4.6.2
```

3. On both nodes, first the standby node, then the primary node, register the DB2 instances with Tivoli System Automation using the command shown in Example 6-10. The command is found in the /opt/IBM/db2/V8.1/ha/salinux directory.

Example 6-10 Running the database registration script

```
[root@prague]# ./regdb2salin -a db2inst1 -r -s
[root@edinburgh]# ./regdb2salin -a db2inst1 -r -s
```

4. Verify that the resource groups are registered and online by issuing the following command:

```
# /usr/sbin/rsct/sapolicies/bin/getstatus
```

You see an output similar to Example 6-11.

Example 6-11 Output of getstatus command

```
-- Resource Groups and Resources --
              Group Name                    Resources
              ----------                    ---------
              db2_db2inst1_edinburgh_0-rg  db2_db2inst1_edinburgh_0-rs
              db2_db2inst1_prague_0-rg    db2_db2inst1_prague_0-rs
-- Resources --
        Resource Name           Node Name              State
        -------------           ---------              -----
db2_db2inst1_edinburgh_0-rs     edinburgh              Online
                          -             -                   -
db2_db2inst1_prague_0-rs           prague              Online
                          -             -                   -
```

5. On the primary node (edinburgh) create a Tivoli System Automation resource group for the HADR pair, and check its status with the commands, as shown in Example 6-12. (9.3.5.90 is the service IP address to which you connect the Tivoli Dynamic Workload Broker application server.) The Tivoli Dynamic Workload Broker creates two databases (TDWB and IBMCDB), so we need to create two resource groups.

   Example 6-12 Creating resource groups

   ```
   # ./reghadrsalin -a db2inst1 -b db2inst1 -d TDWB
   # ./reghadrsalin -a db2inst1 -b db2inst1 -d IBMCDB
   ```

6. Create a service dependency relationship between the HADR database and the IP interfaces of the nodes. Before you configure these relationships, stop HADR (as root on both nodes). See Example 6-13.

   Example 6-13

   ```
   # chrg -o offline db2hadr_TDWB-rg
   # chrg -o offline db2hadr_IBMCDB-rg
   ```

   You can use the **getstatus** command until the HADR resources change to offline.

7. Create the Automation relationships between the HADR application, the service IP address, and the network equivalency (as root on both nodes), as shown in Example 6-14. This ensures that the resources start and stop in the correct order.

Example 6-14 Making dependency relationship

```
# mkrel -p dependsOn -S IBM.Application:db2hadr_TDWB-rs \
   -G IBM.ServiceIP:db2_ip-rs tdwbdb_do_ip
# mkrel -p dependsOn -S IBM.Application:db2hadr_IBMCDB-rs \
   -G IBM.ServiceIP:db2_ip-rs ibmcdbdb_do_ip

# lsrel
Displaying Managed Relations :

Name            Class:Resource:Node[Source]       ResourceGroup[Source]
tdwbdb_do_ip    IBM.Application:db2hadr_TDWB-rs   db2hadr_TDWB-rg
ibmcdbdb_do_ip  IBM.Application:db2hadr_IBMCDB-rs db2hadr_IBMCDB-rg
iscip_do_equ    IBM.ServiceIP:isc-ip              isc_ip-rg
brokerip_do_equ IBM.ServiceIP:broker-ip           broker_ip-rg
```

8. Turn the HADR application back on with the command shown in Example 6-15.

Example 6-15 Turning the HADR application back on

```
# chrg -o online db2hadr_TDWB-rg
# chrg -o online db2hadr_IBMCDB-rg
```

9. Check the status of the resources with the **`getstatus`** command. Now you have to tell DB2 that the Tivoli Dynamic Workload Broker database lives on the service IP:

```
%db2 CATALOG TCPIP NODE LBNODE REMOTE 9.3.5.90 SERVER 50000
```

6.4 Installing and configuring WebSphere Application Server

This section describes the steps needed to install and configure IBM WebSphere Application Server V6.0 and configure on a cluster using Tivoli System Automation. IBM WebSphere Application Server has to be installed and patched to the correct level on both sides of the cluster before installing Tivoli Dynamic Workload Broker.

6.4.1 Installing WebSphere Application Server

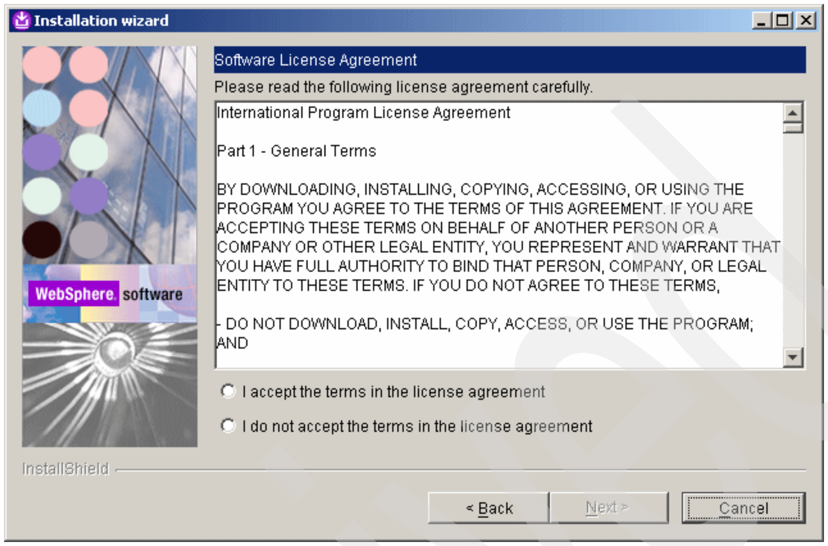
To install IBM WebSphere Application Server on both nodes, perform the following steps:

1. Before installing, identify the installation directory. This item is required when running the installation script.
2. Log in to the node where you want to install the WebSphere Application Server, as a root user.
3. Change directory to the install media and type:

```
# install
```

4. Providing that you have the display parameters set correctly, the install wizard displays a welcome screen. Read the welcome information and click **Next**.

5. Read the license agreement, select the acceptance radio button, and click **Next**, as seen in Figure 6-8.



Software License Agreement

Please read the following license agreement carefully.

International Program License Agreement

Part 1 - General Terms

BY DOWNLOADING, INSTALLING, COPYING, ACCESSING, OR USING THE PROGRAM YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCEPTING THESE TERMS ON BEHALF OF ANOTHER PERSON OR A COMPANY OR OTHER LEGAL ENTITY, YOU REPRESENT AND WARRANT THAT YOU HAVE FULL AUTHORITY TO BIND THAT PERSON, COMPANY, OR LEGAL ENTITY TO THESE TERMS. IF YOU DO NOT AGREE TO THESE TERMS,

- DO NOT DOWNLOAD, INSTALL, COPY, ACCESS, OR USE THE PROGRAM; AND

○ I accept the terms in the license agreement

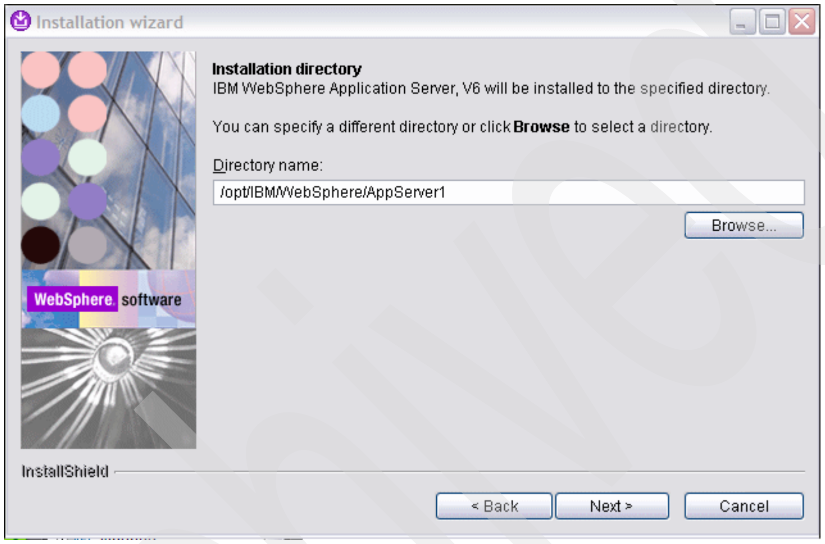
○ I do not accept the terms in the license agreement

Figure 6-8 WAS install: license agreement screen

6. You are then presented with the System prerequisites check screen. Read this and if the last line states `Your system completed the prerequisites check successfully` then click **Next**.

7. You are now presented with the Installation directory screen. Yype in the directory where you would like the installation to reside, then click **Next**, as seen in Figure 6-9.

Tip: Keep the installation directory the same for both systems in the cluster.



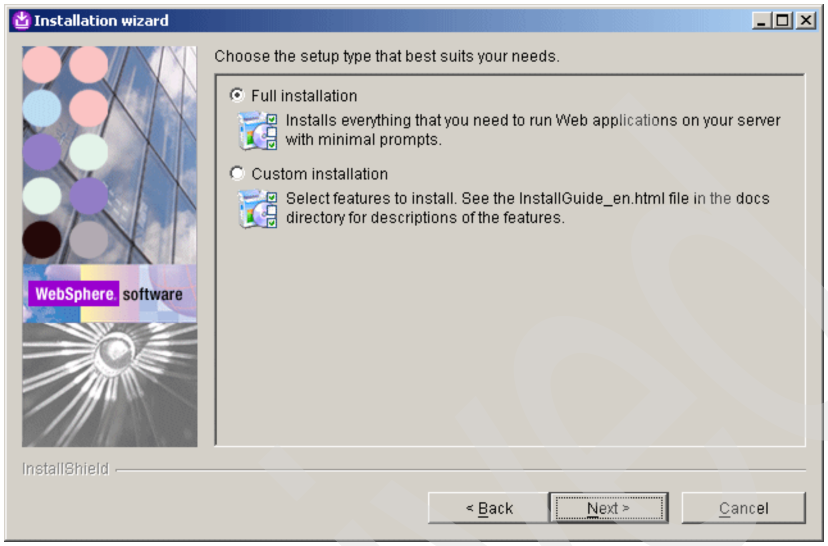
Installation directory
IBM WebSphere Application Server, V6 will be installed to the specified directory.

You can specify a different directory or click **Browse** to select a directory.

Directory name:
/opt/IBM/WebSphere/AppServer1

Figure 6-9 WAS installation: Installation directory screen

8. Choose the type of installation (ether a full or custom installation), select **Full installation**, and click **Next**, as seen in Figure 6-10.



Choose the setup type that best suits your needs.

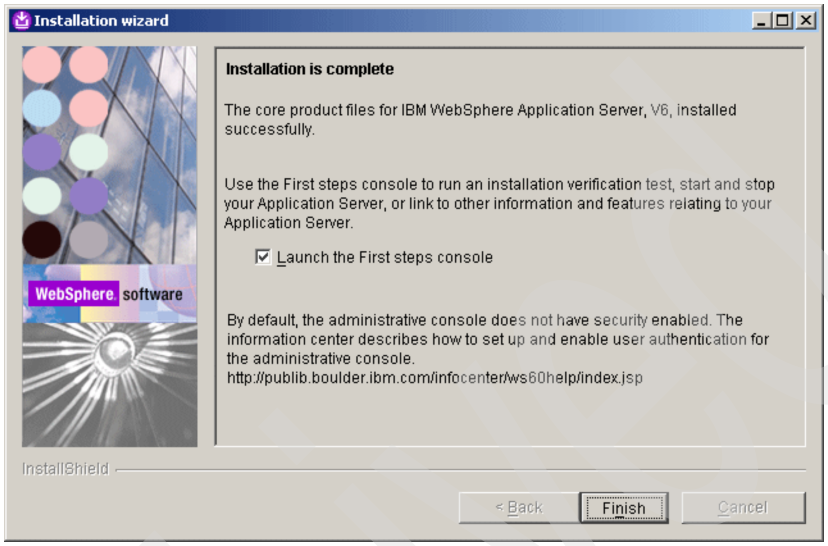
(●) Full installation
Installs everything that you need to run Web applications on your server with minimal prompts.

( ) Custom installation
Select features to install. See the InstallGuide_en.html file in the docs directory for descriptions of the features.

Figure 6-10 WAS install: type of install

9. The next screen is a summary screen. Read this screen to check that it will install every thing you wish, then click **Next**.

10.The next screen is a installation progress bar. Wait until it returns with `Installation Complete`, check the **Launch the first steps console** radio button, and click **Finish**, as seen in Figure 6-11.



Installation is complete

The core product files for IBM WebSphere Application Server, V6, installed successfully.

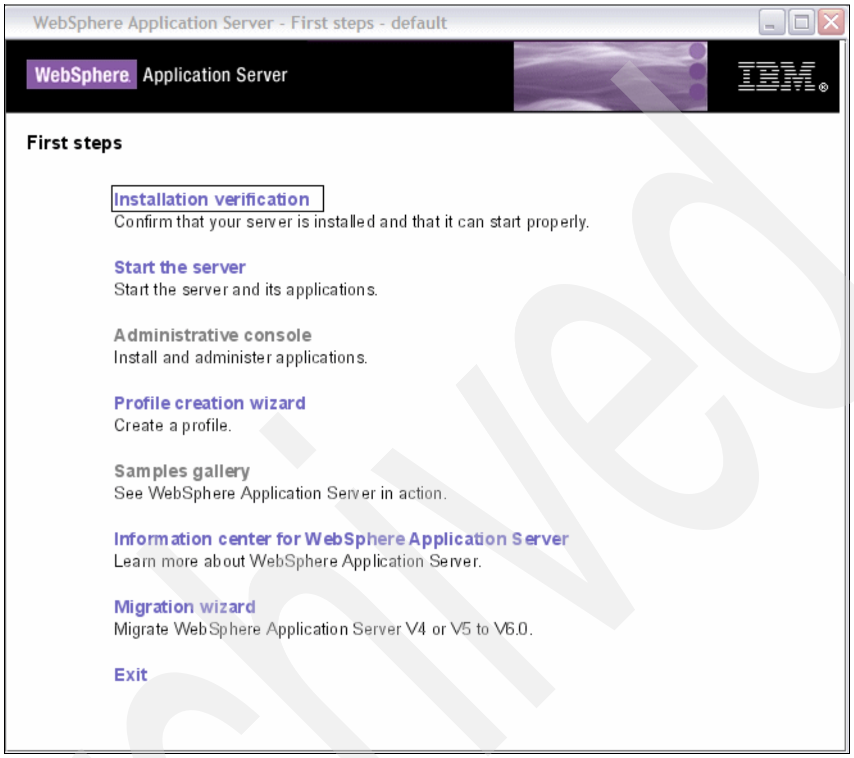
Use the First steps console to run an installation verification test, start and stop your Application Server, or link to other information and features relating to your Application Server.

☑ Launch the First steps console

By default, the administrative console does not have security enabled. The information center describes how to set up and enable user authentication for the administrative console.
http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp

Figure 6-11 WAS Install: Installation complete screen

11. The next screen is the WebSphere Application Server first steps. Click **Installation verification** to see whether the server is installed correctly, as seen in Figure 6-12.


WebSphere Application Server - First steps - default

WebSphere Application Server

First steps

Installation verification
Confirm that your server is installed and that it can start properly.

Start the server
Start the server and its applications.

Administrative console
Install and administer applications.

Profile creation wizard
Create a profile.

Samples gallery
See WebSphere Application Server in action.

Information center for WebSphere Application Server
Learn more about WebSphere Application Server.

Migration wizard
Migrate WebSphere Application Server V4 or V5 to V6.0.

Exit

Figure 6-12 WAS installation: installation verification

12. Once you have selected the installation verification, a new screen appears. Check this screen for any errors once the last line reads `Installation Verification complete`.

13. Close the verification screen and click **Exit** on the First steps screen. This finishes the base install of WebSphere Application Server.

14. Repeat steps 1 through to 13 on the second node using the same settings as the first node.

We now have the WebSphere Application Server installed on both sides of the cluster. The next set is to install the patch, as described next.

6.4.2 Installing WebSphere Application Server patch

WebSphere Application Server, Version 6.0, 32-bit version, with Refresh Pack 2 and Fix Pack 11 is a prerequisite for the successful installation of Tivoli Dynamic Workload Broker. Below are the main steps needed to install the WebSphere Application Server patch, but for more details refer to the readme file.

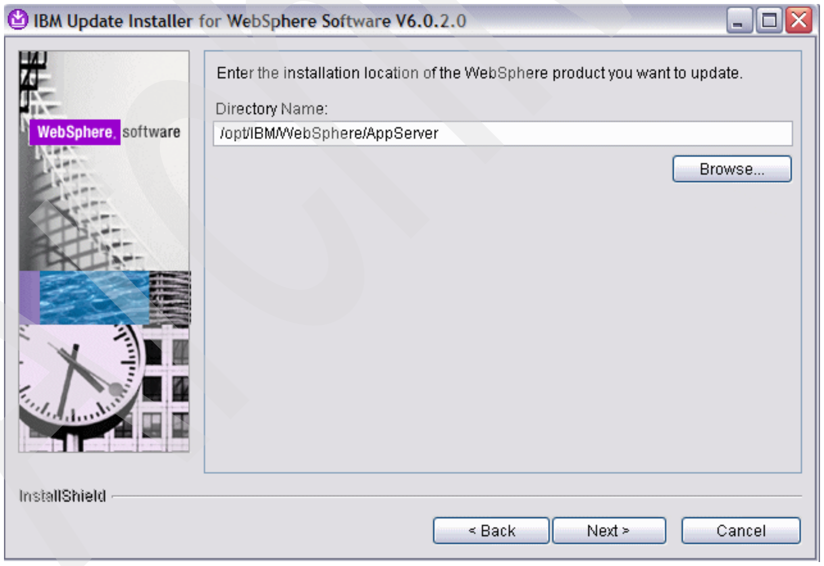
1. Log in as root.
2. Change the directory to where the WebSphere Application Server is installed:

   ```
   #cd /opt/IBM/WebSphere/AppServer
   ```

3. Extract the tar file in this directory:

   ```
   #tar xvf 6.0.2-WS-WAS-LinuxX32-FP00000011.tar
   ```

4. Extracting the tar file creates a directory called updateinstaller. Change to this directory and run the executable *update*.
5. First a welcome screen appears. Read this and then click **Next**.
6. The next screen asks for the location of the WebSphere product that you wish to update. Type in `/opt/IBM/WebSphere/AppServer` or the location of your WebSphere Application Server installation and click **Next**, as shown in Figure 6-13.



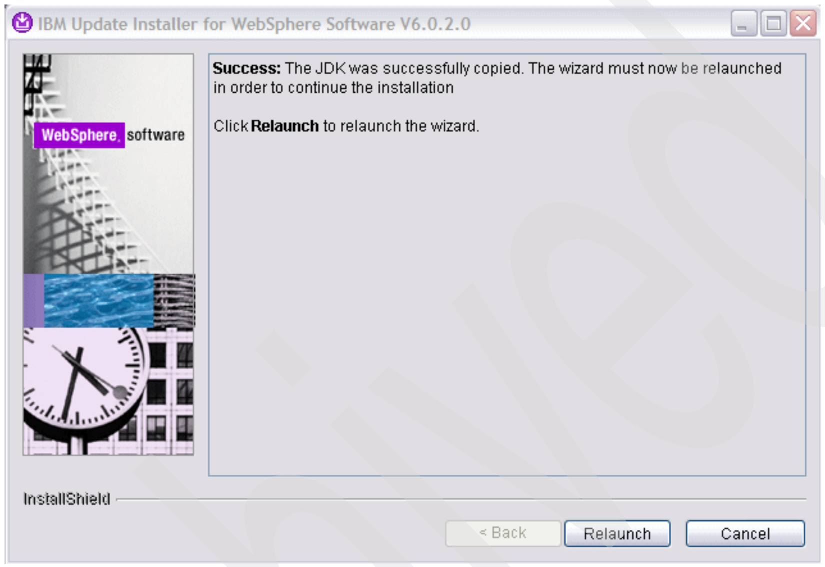
Enter the installation location of the WebSphere product you want to update.

Directory Name: /opt/IBM/WebSphere/AppServer

Figure 6-13 WAS FP 02 Installation: Location of WebSphere Application Server

7. The next screen asks you whether you wish to install or uninstall a maintenance package. Select **Install maintenance package** and click **Next**.

8. On the next screen type in the name of the maintenance package to install, then click **Next**.
9. On the next screen you will see that the JDK™ was successfully copied and the install wizard needs to be relaunched. Click **Relaunch** to finish the installation, as shown in Figure 6-14.



Success: The JDK was successfully copied. The wizard must now be relaunched in order to continue the installation

Click **Relaunch** to relaunch the wizard.

Figure 6-14 WAS FP 02 Installation: relaunch Installer wizard

10. You are then presented with an information screen as to which application server will be upgraded and with which maintenance package. Check this and when satisfied click **Next**.

    The next screen shows the component being backed up.
11. You are presented with a success screen stating that the product was successfully upgraded. Click **Finish**.
12. Repeat the steps 1 through to 12 on the second node using the same settings as the first node.

We now have WebSphere Application Server FP 02 installed on both sides of the cluster. The next step is to configure WebSphere Application Server on the Tivoli System Automation, as described in 6.4.3, "Set up WebSphere Application Server on Tivoli System Automation" on page 271.

6.4.3 Set up WebSphere Application Server on Tivoli System Automation

As Tivoli Dynamic Workload Broker is an application that runs on the WebSphere Application Server, we do not have to set up WebSphere Application Server on Tivoli System Automation, but if you would like to do this then refer to Chapter 10 of *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688.

6.5 Installing and configuring Tivoli Dynamic Workload Broker

This section describes the steps needed to install and configure Tivoli Dynamic Workload Broker and configure on a cluster using Tivoli System Automation. The Tivoli Dynamic Workload Broker server has to be installed and patched to the correct level on both sides of the cluster.

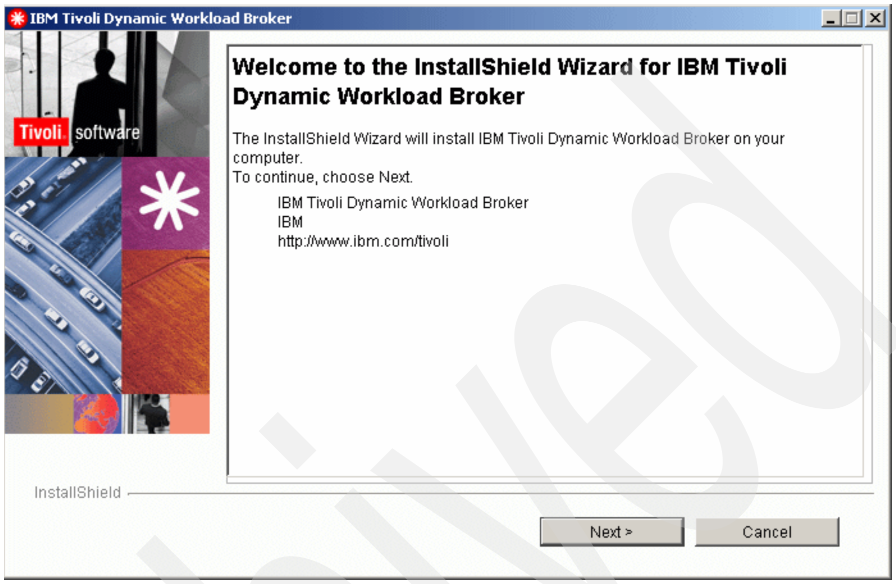
6.5.1 Installing Tivoli Dynamic Workload Broker

This section describes the steps needed to install the Tivoli Dynamic Workload Broker server. For a full installation description refer to *IBM Tivoli Dynamic Workload Broker Installation and Configuration,* SC32-2282. On both nodes perform the following steps:

1. Before installing, identify the installation directory. This item is required when running the installation script.
2. Log in to the node where you want to install the Tivoli Dynamic Workload Broker server, as a root user.
3. Change the directory to the install media and type:

```
# setuplinux.bin
```

4. Provided that you have the display parameters set correctly, the install wizard displays a welcome screen. Read the welcome information and click **Next**, as shown in Figure 6-15.



Welcome to the InstallShield Wizard for IBM Tivoli Dynamic Workload Broker

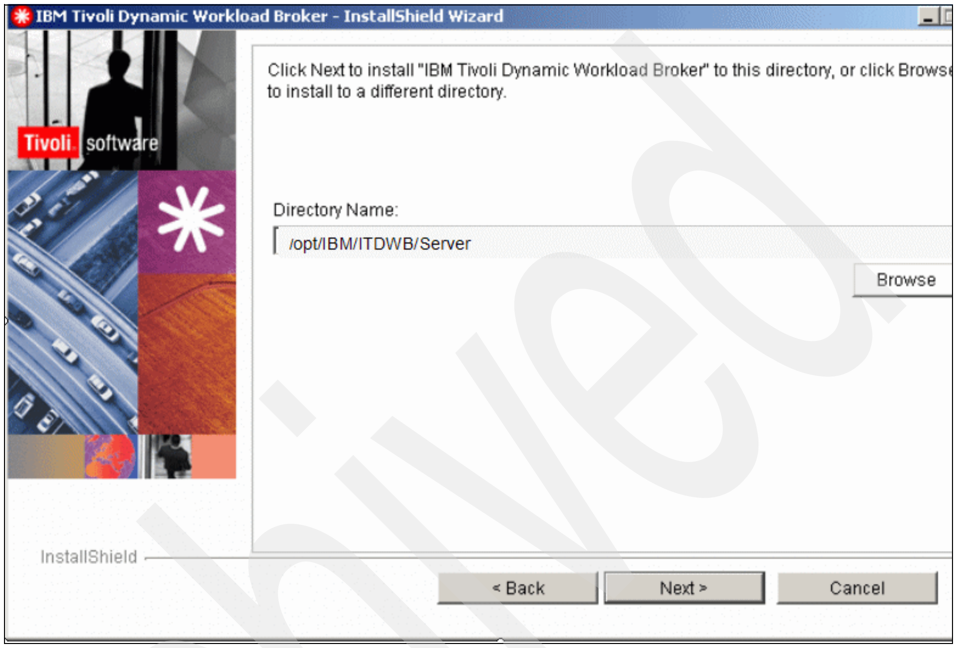
The InstallShield Wizard will install IBM Tivoli Dynamic Workload Broker on your computer.
To continue, choose Next.

IBM Tivoli Dynamic Workload Broker
IBM
http://www.ibm.com/tivoli

Figure 6-15 Tivoli Dynamic Workload Broker Install welcome screen

5. The license agreement window is displayed. Read the license agreement. Click **Print** if you want to print a copy of the license agreement. Check **I accept** for both the IBM and the non-IBM terms, then click **Next**.

6. The installation directory window is displayed. If you want to change the suggested directory on the window, click **Browse** and select a different drive or directory. Otherwise, click **Next** to accept the directory, as shown in Figure 6-16.



Click Next to install "IBM Tivoli Dynamic Workload Broker" to this directory, or click Browse
to install to a different directory.

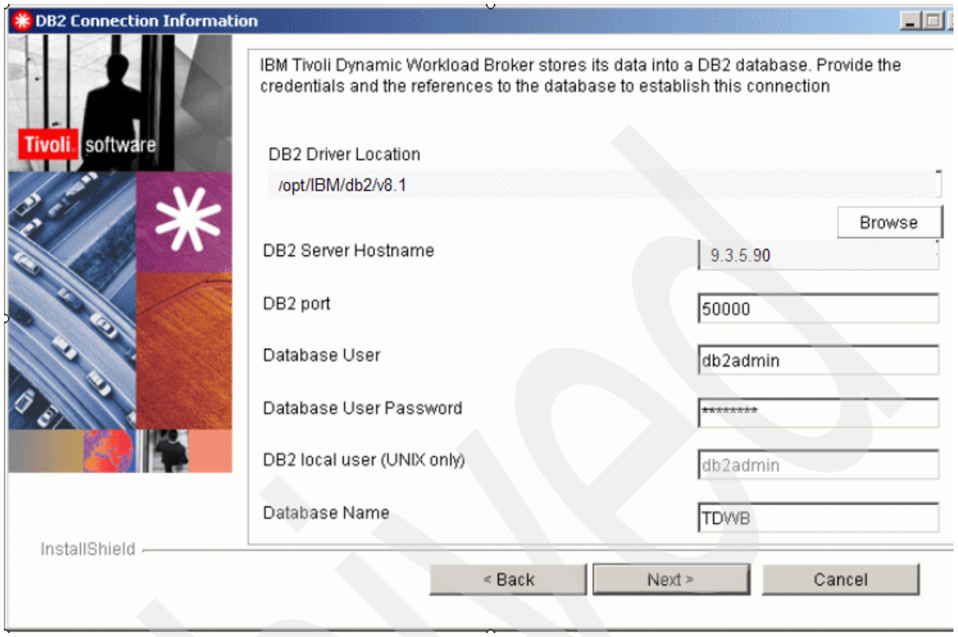
Directory Name:

/opt/IBM/ITDWB/Server

Figure 6-16 Tivoli Dynamic Workload Broker Install directory screen

7. Select the type of installation (either Typical or Custom). For this installation we selected Typical. Once checked click **Next**.

8. You will then be asked about the DB2 installation, as shown in Figure 6-17.



IBM Tivoli Dynamic Workload Broker stores its data into a DB2 database. Provide the credentials and the references to the database to establish this connection

DB2 Driver Location	/opt/IBM/db2/v8.1
DB2 Server Hostname	9.3.5.90
DB2 port	50000
Database User	db2admin
Database User Password	********
DB2 local user (UNIX only)	db2admin
Database Name	TDWB

Figure 6-17 DB2 location information

All of the information displayed in this window is information about the prerequisite DB2 database, except for the database name, which is the name for the Tivoli Dynamic Workload Broker database. Check the following:

– DB2 driver location: The directory where the client or server version of DB2 is installed.

– DB2 server hostname: The name of the host that you are going to connect to DB2. The default is the fully qualified host name of the local computer. This can be either a DB2 server or a DB2 client. For this we must use the floating IP address, and for this installation we used 9.3.5.90.

– DB2 port: The port on which DB2 will listen. This is normally 50000.

– Database user and database user password: The DB2 instance owner and password pair for which an account can be created on this computer. The user ID that you enter here is used for the connection to the DB2 database. You can use the predefined user ID db2admin (Windows) or db2inst1 (UNIX). Alternatively, you can use a new user ID if you require greater security. If you do this, the user ID must exist in the DB2 database. The user must be already configured in DB2 and on the DB2 server.

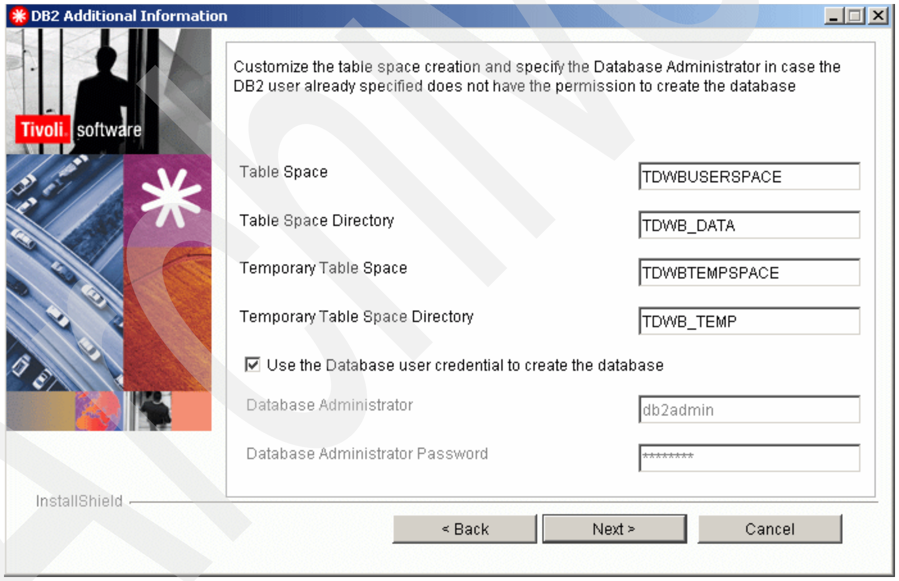
– DB2 local user (UNIX only): The user on the local operating system that is running the DB2 client binaries.

– Database name: The name of the Tivoli Dynamic Workload Broker database. There are restrictions on the length of the database name. For more information, refer to the DB2 manuals.

  The Tivoli Dynamic Workload Broker database will be created on the computer where the DB2 server resides. The default database name TDWB will be used unless you change it. If the database exists, the existing database is used and the Database Name field is not displayed.

Click **Next**. A check is performed to establish a connection with the DB2 database server. If no connection can be made, an error message is displayed. Make sure that the DB2 database is started. When a connection to the DB2 database is established, the next window is displayed.

9. If a Tivoli Dynamic Workload Broker database has not previously been installed, the DB2 additional information window is displayed, as shown in Figure 6-18.



Customize the table space creation and specify the Database Administrator in case the DB2 user already specified does not have the permission to create the database

Table Space	TDWBUSERSPACE
Table Space Directory	TDWB_DATA
Temporary Table Space	TDWBTEMPSPACE
Temporary Table Space Directory	TDWB_TEMP

☑ Use the Database user credential to create the database

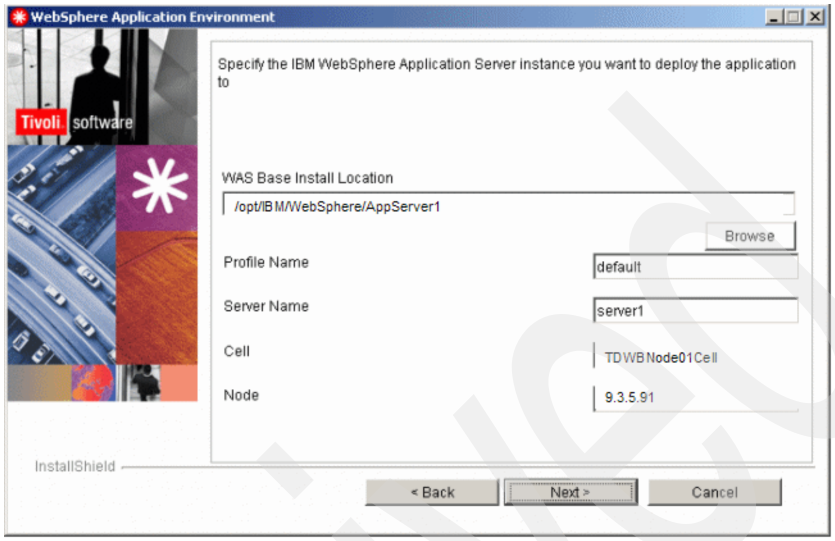
Database Administrator	db2admin
Database Administrator Password	********

Figure 6-18 Additional database information

If required, specify the table space, table space directory, temporary table space, and temporary table space directory. Otherwise leave the default values unchanged. In our installation we left all of the options as default.

Click **Next**.

10. The WebSphere Application environment window is displayed, as shown in Figure 6-19.



Specify the IBM WebSphere Application Server instance you want to deploy the application to

WAS Base Install Location: /opt/IBM/WebSphere/AppServer1

Profile Name: default

Server Name: server1

Cell: TDWBNode01Cell

Node: 9.3.5.91

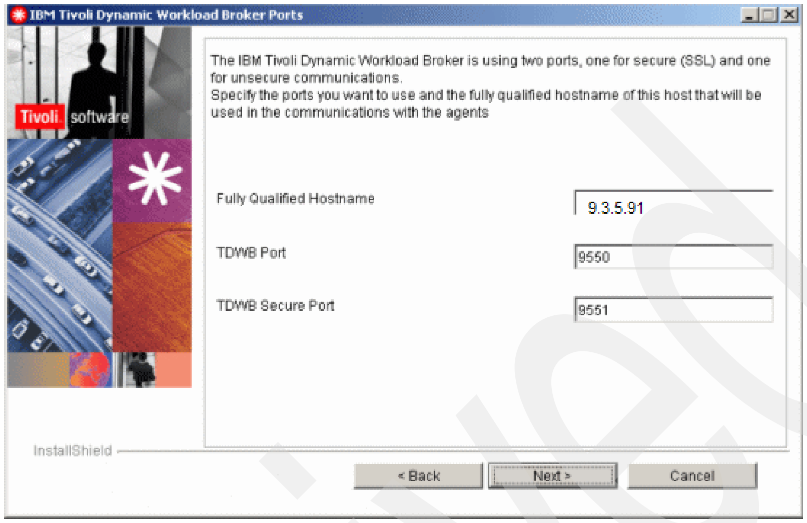
Figure 6-19 WebSphere Application environment window

Check the following WebSphere information displayed in the window:

- Base install location: The directory where WebSphere Application Server is installed.
- Profile name: The existing name of a WebSphere Application Server profile. The profile used here must exist.
- Server name: The existing name of the WebSphere Application Server.
- Cell: The cell name. This is automatically discovered if a valid directory is specified for the base install location.
- Node: The WebSphere node name. This is automatically discovered if a valid directory is specified for the base install location. For our installation we used the floating IP address of 9.3.5.91.

Click **Next** to continue.

11. The Tivoli Dynamic Workload Broker Ports window is displayed, as shown in Figure 6-20.


IBM Tivoli Dynamic Workload Broker Ports

The IBM Tivoli Dynamic Workload Broker is using two ports, one for secure (SSL) and one for unsecure communications.
Specify the ports you want to use and the fully qualified hostname of this host that will be used in the communications with the agents

Fully Qualified Hostname: 9.3.5.91

TDWB Port: 9550

TDWB Secure Port: 9551

Figure 6-20 IBM Tivoli Dynamic Workload Broker Ports

This contains the values for the two ports used by Tivoli Dynamic Workload Broker. The TDWB port value is used for unsecure communications. The TDWB secure port value is used for secure (SSL) communications. The default values are provided. You can change the values if you need to do so. For our example we changed the value of the host name to the IP address 9.3.5.91, as this IP will move with the WebSphere Application server between the two nodes.

Click **Next**.

12. The Agent Manager information window is displayed. For our installation we left all settings as default. If you are happy with the default setting click **Next**.
13. A installation summery window will then be displayed. Read this then click **Next**.
14. An installation progress bar will be displayed, and then the installation completed window will be displayed. Click **Finish** to close the installer.

6.5.2 Setting up Tivoli Dynamic Workload Broker server on Tivoli System Automation

To configure Tivoli System Automation to manage the Tivoli Dynamic Workload Broker, edit the configuration file and run the policy installation script to create the Tivoli System Automation configurations. Place the configurations and start, stop, and monitor scripts for Tivoli Dynamic Workload Broker in /usr/sbin/rsct/sapolicies/tdwb.

We assume that System Automation is already configured and has a domain already set up and working. If you need to set one up then refer to 6.2.2, “Installing and configuring Tivoli System Automation” on page 252.

1. We first make a resource group. A resource group is a container for a set of resources that should be operated on as a single entity. It does not matter on which system this and the following commands are executed.

   ```
   # mkrg broker_ip-rg
   ```

2. We need to create resource definition files for the Tivoli Dynamic Workload Broker server and the IP address to be managed by Tivoli System Automation. Using these examples, create your own definition text files (for example, broker.def), as shown in Example 6-16, and run the following commands to create the resources:

   ```
   # mkrsrc IBM.ServiceIP Name="broker-ip" IPAddress=9.3.5.91 \
   NetMask=255.255.254.0 NodeNameList="{'edinburgh','prague'}"
   ```

   Example 6-16 Creating the resources

   ```
   # IBM_PROLOG_BEGIN_TAG
   # This is an automatically generated prolog.
   #
   #
   #
   # Licensed Materials - Property of IBM
   #
   # (C) COPYRIGHT International Business Machines Corp. 2003,2006
   # All Rights Reserved
   #
   # US Government Users Restricted Rights - Use, duplication or
   # disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
   #
   # IBM_PROLOG_END_TAG
   PersistentResourceAttributes::
   Name="broker-rs"
   StartCommand="/usr/sbin/rsct/sapolicies/was/wasctrl-as start
   /opt/IBM/WebSphere/AppServer 9550 server1"
   StopCommand="/usr/sbin/rsct/sapolicies/was/wasctrl-as stop
   /opt/IBM/WebSphere/AppServer 9550 server1"
   MonitorCommand="/usr/sbin/rsct/sapolicies/was/wasctrl-as status
   /opt/IBM/WebSphere/AppServer 9550 server1"
   MonitorCommandPeriod=19
   ```

```
MonitorCommandTimeout=30
NodeNameList={"edinburgh","prague"}
StartCommandTimeout=300
StopCommandTimeout=180
UserName="root"
RunCommandsSync=1
ResourceType=1
ProtectionMode=1
```

3. We then have to add the resource to the resource group. To do this run the following commands to add the previously created resources to the resource group:

   ```
   # addrgmbr -g broker_ip-rg IBM.ServiceIP:broker-ip
   ```

4. Next we create the equivalencies. Create an equivalency for the network adapters on both servers that we want to use for the IP alias. After creating the equivalencies you can list these by using the command **lsequ**, as shown in Example 6-17.

   ```
   # mkequ -D "Name like 'eth0'" broker_equ IBM.NetworkInterface
   ```

   Example 6-17 Display equivalencies

   ```
   lsequ
   Displaying Equivalencies :
   broker_equ
   db2_equ
   ```

5. We now need to specify the dependencies. Execute the following command to specify that the IP address is dependant on the equivalence:

   ```
   # mkrel -p dependson -S IBM.ServiceIP:broker-ip -G \
   IBM.Equivalency:broker_equ brokerip_do_equ
   ```

   To display the managed relations use the command **lsrel**, as shown in Example 6-18.

Example 6-18 Displaying managed relations

```
[root@edinburgh db2]# lsrel
Displaying Managed Relations :

Name             Class:Resource:Node[Source]      ResourceGroup[Source]
tdwbdb_do_ip     IBM.Application:db2hadr_TDWB-rs   db2hadr_TDWB-rg
ibmcdbdb_do_ip   IBM.Application:db2hadr_IBMCDB-rs db2hadr_IBMCDB-rg
brokerip_do_equ  IBM.ServiceIP:broker-ip           broker_ip-rg
```

6.6 Testing the environment

To test a failover manually, you can move the Tivoli System Automation resources for the Tivoli Dynamic Workload Broker server to the other node with the commands shown in Example 6-19.

Example 6-19 Manually moving the server

```
# rgreq -o move db2hadr_TDWB-rg
# rgreq -o move broker_ip-rg
```

Use the **getstatus** command to see the outcome and that the resource group is now online on the other side.

Performance optimization

The performance of Tivoli Dynamic Workload Broker depends largely on how it is configured in the customer environment. After the initial installation, the product is ready to be up and running with its configuration parameters set with some default values. There is not an absolute *best* configuration, but configuration for best performance varies based on the environment in which the product runs and the expectations from the product. In this chapter we describe how to configure the Tivoli Dynamic Workload Broker and what the best practices are to tune it for performance optimization.

The following are discussed in this chapter:

7.1 Configuration parameters

The configuration of the Tivoli Dynamic Workload Broker, for both server and agent, is essentially stored in properties files (for the server also in WebSphere Application Server configuration files). These properties files are created at installation time, so it is not required to modify them once the product is installed. The Tivoli Dynamic Workload Broker is configured with some of the values specified during the installation process, as well as the URL location of the agent manager, but for many others, default values are used.

All of the parameters influencing the performance of the product are set to default values. These values are the best values for a general use of the product but not necessarily the best values for your environment. This is because performances can depend largely on many factors as well as the number of agents, expected jobs running in a day, the maximum number of jobs running concurrently, the CPU on the server, the network bandwidth, and so on. It would be a good practise to tune the configuration on the server and on the agents to run the IBM Tivoli Dynamic Workload Broker using the *correct* configuration.

The server can be configured by modifying some of the parameters defined in the properties files, located under the directory <install location>/config, *JobDispatcherConfig.properties* and *ResourceAdvisorConfig.properties*.

The agent can be configured by modifying some of the parameters defined in the properties file, *ResourceAdvisorAgentConfig.properties*, located under the directory <install location>/ep/runtime/agent/subagents, and *JobExecutionAgentConfig.properties*, located under the directory <install location>/ep/runtime/agent/subagents/JobExecutionAgent.

The parameters can be easily modified by simply editing the properties file in a flat text editor, or, for the JobExecutionAgentConfig, using the command-line interface on the agent.

7.2 Performance configuration parameters on server

We start by discussing performance configuration parameters on the server.

7.2.1 Job Dispatcher

The JobDispatcherConfig defines the behavior of the server in relation to the dispatching of jobs. Each of these properties defines a rule on how the server processes jobs, manages their status changes, and saves them in the job repository database.

The properties are reloaded by the server at an interval time defined with the FailQinterval parameter and each of them must be a integer number included in the interval of values defined in the Range column. If not so, the default value is taken. Also, in the Runtime changes column, *no* indicates that the WebSphere application server must be restarted to have the new value be effective. Table 7-1 shows the Job Dispatcher properties parameters.

Table 7-1 Job Dispatcher properties

Property	**Definition**	**Default**	**Range**	**Run-time change**
FailQInterval	Specifies the numbers of seconds for retrying the operation after the following Job Dispatcher failures: ▶ Client notification ▶ Allocation, Reallocate, Cancel Allocation requests to Resource Advisor ▶ Any operation on DB failed for connectivity reasons	30 seconds	5–600	Yes
MaxNotificationCount	Specifies the maximum number of retries after a Job Dispatcher failure.	1440	20–28800	Yes
MoveHistoryDataFrequencyInMins	Specifies how often job data must be checked: ▶ To be moved to the archive tables in the Job Repository database ▶ To be removed from the archive database tables	60 minutes	5–14400	Yes
SuccessfulJobsMaxAge	Specifies how long jobs completed successfully or cancelled must be kept in the Job Repository database before being archived.	24 hours	1–8784	Yes
UnsuccessfulJobsMaxAge	Specifies how long jobs completed unsuccessfully or in unknown status must be kept in the Job Repository database before being archived.	72 hours	1–8784	Yes

Property	Definition	Default	Range	Run-time change
ArchivedJobs MaxAge	Specifies how long jobs in the archive database tables must be kept in the Job Repository database before being removed.	168 hours	1–43920	Yes

The Tivoli Dynamic Workload Broker V1.2 introduces a new threading model so that the Job Dispatcher Manager can work with a configurable number of threads to have faster queue processing. Other parameters have been added to the JobDispatcherConfig, as specified in Table 7-2 on page 285.

By default, these parameters are not defined in the JobDispatcherConfig properties file but you can add new rows to it in order set new values, as in Example 7-1.

Example 7-1 Job Dispatcher configuration properties

```
######################################################
# Job Dispatcher Configuration Properties
#
######################################################
# Resource Advisor serivice address
RAEndpointAddress =
http://localhost:9550/RAServiceWS/services/AllocationFactory
# Job Dispatcher serivice address
JDURL=http://localhost:9550
# Numbers of seconds for Job ispatcher failure retry
FailQInterval = 30
# maximum numbers of retry after client notification failure
MaxNotificationCount = 120
# Frequency to trigger job history data move to archive and archive
cleanup
MoveHistoryDataFrequencyInMins = 60
# Max age in hours to keep job successfully terminated or cancelled in
the online tables. Default 0.5 day
SuccessfulJobsMaxAge = 12
# Max age in hours to keep job unsuccessfully terminated or unknown in
the online tables. Default 3 days
UnsuccessfulJobsMaxAge = 72
```

```
# Max age in hours to keep job in the archive tables. Default 1 week
ArchivedJobsMaxAge = 168
#
MaxProcessingWorkers = 20
```

Table 7-2 shows additional Job Dispatcher properties parameters that will be available with Tivoli Dynamic Workload Broker V1.2.

Table 7-2 Job Dispatcher properties - additional properties for TDWB 1.2

Property	**Definition**	**Default**	**Range**	**Run-time change**
MaxProcessing Workers	Specifies how many concurrent threads can be used by Job Dispatcher to process the job status transitions. To get the effective maximum number of threads that can be created you have to multiply this for a factor of 2 + 20 units or 5 +20 units if the Tivoli Workload Scheduler agent is running on top of the Tivoli Dynamic Workload Broker server.	10	1–100	No
HistoryDataCh unk	Specifies the maximum number of jobs moved to archive or removed from there within each transaction when the history data move is triggered.	1000	100–10000	Yes

Queue.actions.<n>	Specifies which actions are processed by which queue number. This parameter allows a finer customization of Job Dispatcher processing queues to be more flexible in finding the optimal configuration.	Queue.actions.0 = CANCEL, CANCEL_ALLOCATION, COMPLETE Queue.actions.1 = ALLOCATION_RECEIVED, REALLOCATE Queue.actions.2 = SUBMITTED, NOTIFICATION, STATUS_UPDATE_FAILED	0–9	Yes
Queue.size.<n>	Specifies the size of each queue. If set to 0, it means that the jobs in the corresponding queue will be not executed.	All equal to MaxProcessing Workers. Only Queue.size.1 = 20	1–100	Yes

7.2.2 Resource Advisor

The ResourceAdivisorConfig defines the behavior of the server in relation to the management of allocating resources.

The properties are reloaded by the server at an interval time defined with the CheckInterval parameter, and each of them must be a integer number included in the range defined in the Range column. If not so, the default value is taken.

Table 7-3 shows the Resource Advisor properties parameters.

Table 7-3 Resource Advisor properties

Property	Definition	Default	Range or format	Run-time change
RaaHeart BeatInterv al	Specifies the time interval within which the Resource Advisor expects a heartbeat signal from the Workload Agent. When this interval expires, the Resource Advisor checks whether the heartbeat signal has been received for each Workload Agent. After a maximum number of consecutive times missing, as specified in MissedHeartBeatCount, the Resource Advisor sets the related computer to unavailable.	367 seconds	15–3600	Yes
MissedHe artBeatCo unt	Specifies the number of missed heartbeat signals after which the computer is listed as not available.	2	1–100	Yes
MaxWaitin gTime	Specifies the maximum time interval a job must wait for a resource to become available. If the interval expires before a resource becomes available, the job status changes to Resource Allocation Failure. This value can be overridden for each specific job by the Maximum Resource Waiting Time parameter defined in the Job Brokering Definition Console.	600 seconds	0–864000	Yes (This applies only to new submitte d jobs.)
CheckInter val	Specifies the minimum time interval within which the Resource Advisor has to wait before re-trying to find matching resources for a job that did not find any resource in the previous time slot.	60 seconds	5–3600	Yes

Property	Definition	Default	Range or format	Run-time change
TimeSlotLength	Specifies the timeslot interval within which the Resource Advisor decides which resources to allocate to each job. Jobs submitted after this interval has expired are considered in a new timeslot.	15 seconds	1–600	Yes
NotifyTimeInterval	Specifies the interval within which the Resource Advisor tries to send notifications on the job status to the Job Dispatcher.	15 seconds	5–600	Yes
MaxNotificationCount	Specifies the maximum number of attempts for the Resource Advisor to send notifications to the Job Dispatcher.	100	1–1200	Yes

The Tivoli Dynamic Workload Broker V1.2 introduces new parameters for configuring the Resource Advisor, as shown in Table 7-4.

By default, these parameters are not defined in the ResourceAdvisorConfig properties file, but you can add new rows to it in order set new values.

Table 7-4 Resource Advisor properties - additional properties for TDWB 1.2

Property	Definition	Default	Range	Run-time change
MaxAllocsInCache	Specifies the maximum number of allocations data the Resource Advisor can hold. When this limit is reached then the Resource Advisor refuses any allocation request from Job Dispatcher.	5000	1–1,000,000	Yes

MaxAllocsPerTimeSlot	Specifies the maximum number of allocations the Resource Advisor can process every time the timeslot expires. If there are more requests in the request queue the Resource Advisor only picks up first MaxAllocsPerTimeSlot requests. The other requests will be processed when the timeslot expires again.	100	1–1,000,000	YES
MaxExtensionCount	Specifies the number of times the Resource Advisor can extend the resource reservation. The default -1 means infinite, that is, a resource reserved through *allocation* remains reserved to a given job until it terminates. Setting a different value causes the Resource Advisor to extend the resource reservation on consumable resources only for the number of timeslots specified.	-1	-1–10,000	YES

7.3 Performance configuration parameters on the agent

There are also a number of parameters on the Tivoli Dynamic Workload Broker agents that can affect the performance.

7.3.1 ResourceAdvisorAgentConfig

The ResourceAdvisorAgentConfig defines the behavior of the agent in relation to the discovering of the resources on the agent and the notification to the server.

The restart of the agent is needed to make the changes effective. Table 7-5 shows the Agent Resource Advisor properties parameters on the agents.

Table 7-5 Agent Resource Advisor properties

Property	Definition	Default
UIMOperatingSystemScanner.ScanIntervalSecs	Specifies how often an operating system scanner is performed on the agent.	30 seconds
UIMComputerSystemScanner.ScanIntervalSecs	Specifies how often a computer system scanner is performed on the agent	30 seconds
UIMFileSystemScanner.ScanIntervalSecs	Specifies how often a file system scanner is performed on the agent.	30 seconds
UIMNetworkScanner.ScanIntervalSecs	Specifies how often a network system scanner is performed on the agent.	30 seconds
ScanOnNotification	Specifies that the scan must be performed immediately before sending the scan data. Supported values are true and false.	false
NotifyToResourceAdvisorIntervalSecs	Specifies the time interval within which the Workload Agent sends a status update on resources to the Resource Advisor. The status update is also used as a heartbeat signal. When this interval expires and no signal is received, the computer is listed as not available.	120 seconds

7.3.2 JobExecutionAgentConfig

The JobExecutionAgentConfig defines the behavior of the agent in relation to jobs execution.

The properties must be an integer number. If not, the default value will be taken. The jobexecutionagent service of the agent command-line interface allows you to modify these parameters in the JobExecutionAgentConfig.properties file. In the agent_installation_directory/ep/runtime/agent directory run the following command:

```
./agentcli.sh jobexecutionagent setproperty propertyname propertyvalue
```

Table 7-6 shows the job execution properties parameters on the agents.

Table 7-6 Job execution properties

Property	**Definition**	**Default**
workmanager.maxjobs	Specifies the maximum number of jobs that can be run concurrently on a resource. When this limit is exceeded, any subsequent jobs assigned to the resource remain in Submitted to Agent status.	40
notifier.maxretries	Specifies the maximum number of attempts for the Workload Agent to notify clients.	480
notifier.retryinterval	Specifies the Workload Agent retry interval between each notification attempt. The time unit is seconds.	60 seconds

7.4 Best practices

In the previous sections we defined what parametes can be changed in order to tune the Tivoli Dynamic Workload Broker. Now we describe how these parameters can influence the performance of the product and what are some of the best practices to follow when changing them.

7.4.1 Server

Table 7-7 discusses best practices for the server parameters.

Table 7-7 Server - Best practices

FailQInterval	Decreasing this number causes the Job Dispatcher to recieve failing situations quicker, but it would require a lot of CPU resources if the jobs usually take long to recover, because the Job Dispatcher will try multiple times to recover the jobs. If the a job requires a long time to run, then it is probably better to set this parameter to higher values than the default ones. One example could be a Tivoli Workload Scheduler agent dealing with a new symphony file. If there are no such kinds of jobs, or they have very little impact on the Tivoli Dynamic Workload Broker, this parameter can be set to lower values.

MaxNotificationCount	The same considerations made for the FailQInterval are also applicable for this parameter, as this should be increased as longer downtimes are expected for jobs requiring long time to terminate.
MoveHistoryDataFrequencyInMins	Increasing this number causes the Job Dispatcher to check less frequently for the job to be moved. The consequence of this is that the volume of the jobs in the Job Repository may be higher and the queries may take more time to complete. Tivoli Dynamic Workload Broker servers with high job throughput may require lower values, as low throughputs may allow higher values. In any case, this parameter does not influence the performance for a great extend.
SuccessfulJobsMaxAge	The considerations made for the MoveHistoryDataFrequencyInMins are also applicable for this parameter.
UnsuccessfulJobsMaxAge	The considerations made for the MoveHistoryDataFrequencyInMins are also applicable for this parameter.
ArchivedJobsMaxAge	The considerations made for the MoveHistoryDataFrequencyInMins are also applicable for this parameter.
MaxProcessingWorkers	Its total value (MaxProcessingWorkers * 2 + 20) should not exceed the maximum number of threads of the Job Dispatcher work manager (the default is set to 50 and can be increased) and the JDBC maximum connections (default set to 100). To check or modify the maximum number of threads of the Job Dispatcher work manager, open the Websphere Application Server administrative console and go to **Resources** → **Asynchronous beans** → **Work managers** → **JobDispatcherWorkManager**. To check or modify the maximum number of JDBC connections, open the administrative console and go to **Resources** → **JDBC providers** → **ITDWBProvider** → **Data sources** → **ITDWBDataSource** → **Connection pools**.
HistoryDataChunk	This parameter is meant to limit the lock escalation in the Job repository when old job data archiving occurs. Consider using lower values if lock escalation occurs and the Tivoli Dynamic Workload Broker server fails in archiving jobs.

RaaHeartBeatInterval	It is advisable to set this parameter to a value higher than the default on a slow network. However, defining a higher value might delay the update on availability status of computer systems. On the other hand, decreasing this number together with the time interval used by the agent to send the heartbeat signal to the Resource Advisor (defined by the parameter NotifyToResourceAdvisorIntervalSecs in the ResourceAdvisorAgentConfig.properties file on the agents) might generate network traffic because updates occur too frequently. Also, the Tivoli Dynamic Workload Broker server will use more CPU to update all the cached data. As a general recommendation, the value defined with this parameter must be consistent with the NotifyToResourceAdvisorIntervalSecs.
MissedHeartBeatCount	On a slow or intermittent network, we recommend setting this parameter to a value higher than the default.
MaxWaitingTime	If you set this parameter to -1, no waiting interval is applied for the jobs. If you set this parameter to 0, the Resource Advisor tries just once to find the matching resources, and if it does not find any resource, then the job goes to ALLOCATION FAILED state. If you increase this value, by default all submitted jobs remain in waiting for a longer time and the Resource Advisor will retry to find matching resources at least every CheckInterval.
CheckInterval	The value defined in this parameter must be consistent with the MaxWaitingTime.
TimeSlotLength	Setting this parameter to a higher value causes the Resource Advisor evaluations to better favor higher priority jobs versus lower priority ones, when some resource is contended (through allocation) by all them. On the other hand, the job resource matching processing takes longer on average and the resource state updates from the agents could be slowed, especially if there is a high server job throughput. Setting this parameter to a lower value causes the Resource Advisor to process the resource matching faster and, in case of many agents with frequent updates, to update the resource repository without delay. If job requirements match many resources, then lower values of this parameter favor a better load balancing. If the major part of the jobs has allocations then it is not indicated to lower this value, as the allocation evaluation requires much processing.
NotifyTimeInterval	It is advisable to set this parameter to be consistent with the notifier.maxretries parameter defined by in the ResourceAdvisorAgentConfig.properties file on agents.
MaxNotificationCount	It is advisable to set this parameter to be consistent with the notifier.retryinterval parameter defined by in the ResourceAdvisorAgentConfig.properties file on agents.

MaxAllocsInCache	Increasing this number causes the Resource Advisor processing a potential higher number of resource reservation data every time slot with consequent processor time usage. This allows the processing of a higher number of jobs. More limited impacts on processing may be obtained if jobs submitted do not use allocations. Decreasing this number causes the Resource Advisor processing a lower number of resource reservation data every time slot, resulting in less processor usage and slower job submission processing.
MaxAllocsPerTimeSlot	Increasing this number causes the Resource Advisor processing a higher number of resource allocation requests data every time slot with consequent processor time usage. Of course, this allows the processing of a higher number of jobs per time slot. Decreasing this number causes the Resource Advisor processing a lower number of resource allocation requests data for each time slot, resulting in a smoother processor usage and slower job submission processing. Moreover, this setting frees-up the Resource Advisor from continuously working on allocations and lets available more time to process resource status update from agents.
MaxExtensionCount	Setting a different value from default (-1) causes the Resource Advisor to extend the resource reservation on consumable resources only for the number of time-slot specified. The effect of this setting is to cause over-allocation in the sense that two or more jobs requesting the same consumable resource can get executed in the same time, even if there is no availability of the requested resource. Nevertheless this allows to free-up resources after a given time letting other jobs to run.

7.4.2 Agent

Table 7-8 discusses best practices for the agent parameters.

Table 7-8 Agent - Best practices

UIMOperatingSystemScanner.ScanIntervalSecs	Increasing this number can improve the performances but can cause the agent to discover changes on the operating system with delay. It is advisable to not set this parameter with a too low value.
UIMComputerSystemScanner.ScanIntervalSecs	Increasing this number can improve the performances but can cause the agent to discover changes on the computer system with delay. It is advisable to not set this parameter with a too low value.
UIMFileSystemScanner.ScanIntervalSecs	Increasing this number can improve the performances but can cause the agent to discover changes on the file system with delay. It is advisable to not set this parameter with a too low value.

UIMNetworkScanner.ScanIntervalSecs	Increasing this number can improve the performances but can cause the agent to discover changes on the network system with delay. It is advisable to not set this parameter with a too low value.
ScanOnNotification	Setting this value to true means that all the scans are performed at each NotifyToResourceAdvisorIntervalSecs interval time, just before sending the notification to the Resource Advisor. All the previous scanning interval parameters will be ignored.
NotifyToResourceAdvisorIntervalSecs	On a slow network, it is advisable to set this parameter to a higher value. The value defined in this parameter should be consistent with the RaaHeartBeatInterval parameter defined in the ResourceAdvisorConfig.properties file on the IBM Tivoli Dynamic Workload Broker server, which defines the time interval within which the Resource Advisor expects a heartbeat signal from the agent.
workmanager.maxjobs	It is recommended to decrease this number if the agent is running on a slow system to limit consumption of resources. Anyway, if the value is too low, an higher number of jobs can remain in to a SUBMITTED state causing an higher number of allocated resources that fill the cache on the Resource Advisor, decreasing performance on server.
notifier.maxretries	Increasing this value might slightly increase network traffic, but this is suggested if the IBM Tivoli Dynamic Workload Broker server runs into a cluster environment. If, for any reason, primary server can not be reached by the agents, an higher value for this parameter allows to the backup server to be notified on jobs' status if it become available in a time shorter then notifier.maxretries * notifier.retryinterval (default is 8 hours). If this time interval is exceeded, the jobs will remain in RUNNING state, up the agents are not restarted.
notifier.retryinterval	Increasing this value might slow the updating of information on the server but the same considerations for notifier.maretries are valid also for this parameter.

7.4.3 A simple scenario for this book

As part of this book, wee run some simple tests to verify how modifying some of the configuration parameters can change the performance of the Tivoli Dynamic Workload Broker server. The environment we used is the following:

- Server: Windows 2003, 4 GB RAM
- Five Agents: three agents Windows, two agents Linux

Three jobs were defined, one running only on Windows platforms, one on Linux and one on AIX.

Jobs have been submitted continuously for 5 minutes (at a rate of about 100 jobs per minute) through the command-line interface. After the 5 minutes of jobs' submission, Resource Advisor continued to work for matching resources for the AIX job, which was still in the WAITING FOR ALLOCATION status.

The percentage of the CPU utilization by the java process running the WebSphere application Server has been measured.

The following parameters on the Resource Advisor have been changed:

- TimeSlotLenght (default 15 sec, 1 sec, 600 sec)
- MaxWaitingTime (default 600 sec, 0 sec, 70 sec)
- CheckInterval (default 60 sec, 1 sec, 200 sec)

Figure 7-1 shows the results for the scenario that tests the TimeSlotLenght.

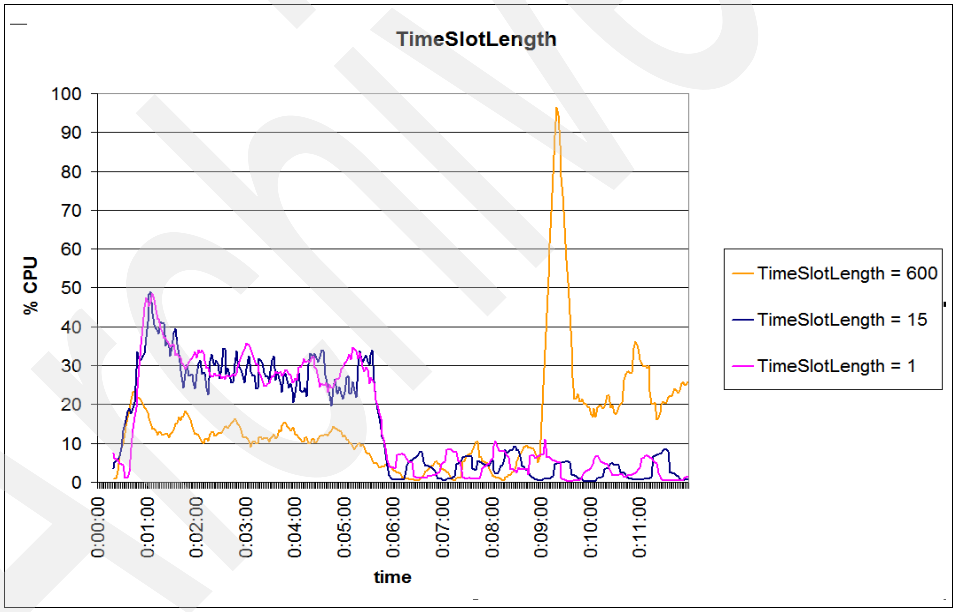


Figure 7-1 Test on TimeSlotLength

When we set a very high value for the TimeSlotChange (for example, 600 sec), we experienced how CPU utilization is effected during the first part of the test, the first 5 minutes, during the jobs' submission. The CPU utilization went down from about 35% (default value) to about %15. This is because a very high value for the TimeSlotChange causes the Resource Advisor that performs the resource matching and allocate resources, not to work for longer periods. The CPU was so occupied only by the Job Dispatcher. The side effect was that, as the TimeSlotChange interval time elapsed, there has been a spike of the CPU utilization due to the Resource Advisor that had to process all the resource allocation requests that were not resolved before. After the spike, the CPU utilization kept to be quite high (about 25%) because the Job Dispatcher continued to process jobs and finally completed them.

Generally, performance depends how the jobs have been defined. The more the submitted jobs have requirements and allocations and the more the Resource Advisor needs processing. Also the number of the agents is a key factor on performance. When there are more agents are in the environment, the Resource Advisor needs more processing for the resource allocation.

The other two tests did not give us significant results. Modifying the values for the MaxWaitingTime and CheckInterval, as done in the tests, did not change performance on the server significantly. The curves obtained are not very different from the curve `TimeSlotChange = 15` in Figure 7-1 on page 296.

7.5 Scalability tests

Some scalability tests have been executed on Tivoli Dynamic Workload Broker V1.1. The results are relative to a specific scenario and to a specific environment, and that there is no guarantee that you will obtain the same results in your environment because many factors can influence performance, as well as the network bandwidth and, most importantly the type of jobs you are running on the agents.

Our environment was:

- Server: AIX 5.3, 4 CPUs, 4 GB RAM, 4 GB for swap
- Agent Manager: running on the same WebSphere Application Server hosting the Tivoli Dynamic Workload Broker server.
- DB2: V8.2 running on the same machine where the Tivoli Dynamic Workload Broker server runs. Two separated disks dedicated to the Tivoli Dynamic Workload Broker server and Agent Manager databases.
- WebSphere Application Server: V6.0.2.11 with heap size to 2 GB.

600 agents connected to the server.

A Tivoli Dynamic Workload Scheduler V8.3 environment has been used to run the test. A variable number of simple jobs (running **ping** command) combined into a single job stream has been submitted to a Tivoli Dynamic Workload agent running on a Tivoli Dynamic Workload Broker server. This routed the jobs to the

Tivoli Dynamic Workload Broker agents. The number of jobs in the job stream has been varied opportunelly.

It has been verified that the job throughput can reach and exceed about 50 jobs for minute (which means about 72,000 jobs per day).

7.5.1 Scenario for the Tivoli Dynamic Workload Broker V1.2

Some other tests has been executed in to an environment upgraded to version V1.2. The new release provides some enhancement of performances. Here is the scenario.

- Server: RedHat Enterperise Linux AS 4, 4CPUs, 4 GB RAM.
- Agent Manager: running on the same WebSphere Application Server hosting the Tivoli Dynamic Workload Broker server.
- DB2: v9.1 server running on the a remote machine, RedHat Enterperise Linux AS 4, 4CPUs, 2 GB RAM.
- WebSphere Application Server: V6.1.5.

5 to 480 agents connected to the server.

As for the test for Tivoli Dynamic Workload Broker V1.1, a Tivoli Dynamic Workload Schduler V8.3 environment has been used to run the test. Very simple jobs (running the `ls` command) have been submitted to a Tivoli Dynamic Workload agent running on the Tivoli Dynamic Workload Broker server with a variable rate (from 50 to 200 jobs per minute). Figure 7-2 summarizes the results.

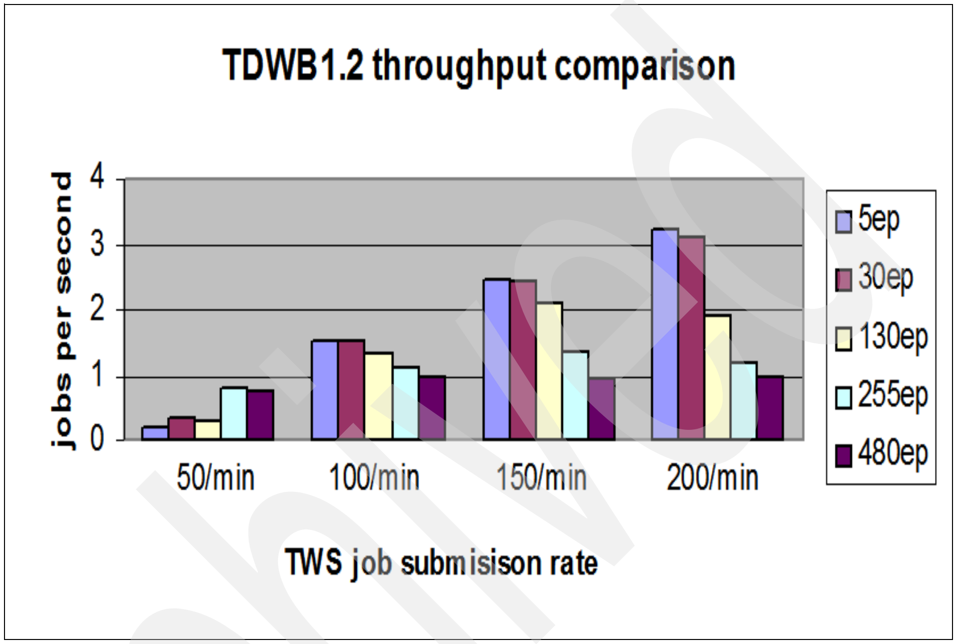


Figure 7-2 Test performance results for ITDWB 1.2

The throughput has been determined on how many of the submitted jobs complete in to a successful state. With a rate of 100 jobs per minute (means about 140,000 jobs for day) almost all jobs have been successful if the number of agents is about 100. With lower rate (about 50 jobs per minute, 72,000 per day) all the jobs complete even with a larger number of agents.

8

Integration with other IBM Tivoli products

Tivoli Dynamic Workload Broker is a powerful tool for distributing jobs across available resources. It has many useful built-in capabilities for managing the workload in production. However, additional efficiency can be achieved by integrating Tivoli Dynamic Workload Broker with other IBM Tivoli Products.

The idea of IBM Tivoli software family is based on the scalability and interoperability of IBM Tivoli products. Integrated IBM Tivoli products working together offer a comprehensive end-to-end solution across the whole IT environment.

In this chapter we describe the mechanism of how Tivoli Dynamic Workload Broker can be integrated with other Tivoli products and thus extend its functionality.

8.1 Our Tivoli Dynamic Workload Broker integration environment

Table 8-1 shows the lab environment that we used for the Tivoli Dynamic Workload Broker integration scenarios. The following sections give you details of these integration scenarios.

Table 8-1 Our lab environment of the Tivoli Dynamic Workload Broker integration scenarios

Host name	OS	Software installed
athens	Windows 2003	► Tivoli Dynamic Workload Broker server V1.1 ► Tivoli Workload Scheduler V8.3 ► Tivoli Entperise Portal V6.1 ► Tivoli Monitoring V6.1
paris	AIX V5.3	Tivoli Provisioning Manager V5.1
zurich	Suse Linux V9.0	Tivoli Change and Configuration Management Database V1.1 server
nice	Windows 2003	Tivoli Dynamic Workload Broker V1.1 Windows agent
oslo	Suse Linux V9.0	Tivoli Dynamic Workload Broker V1.1 Linux agent
cairo	Windows 2003	Tivoli Dynamic Workload Broker V1.1 Windows agent

The following sections give you details of these integration scenarios.

8.2 Integration with IBM Tivoli Change and Configuration Management Database (CCMDB)

As technology becomes intertwined with day-to-day business functions, technology-centric IT management practices are evolving into a more business-focused management of IT services. The IBM IT Service Management strategy enables you to align business insight and innovative technology.

At the core of the IBM IT Service Management strategy is IBM Tivoli Change and Configuration Management Database (Tivoli Change and Configuration Management Database, or CCMDB).

Tivoli Change and Configuration Management Database has native discovery capabilities that an organization can use to obtain a detailed understanding of its supporting infrastructure, including down to layer-2 network devices, storage devices, cross-tier dependencies, and run-time configuration. Tivoli Change and Configuration Management Database provides detailed maps of business applications and their relationships.

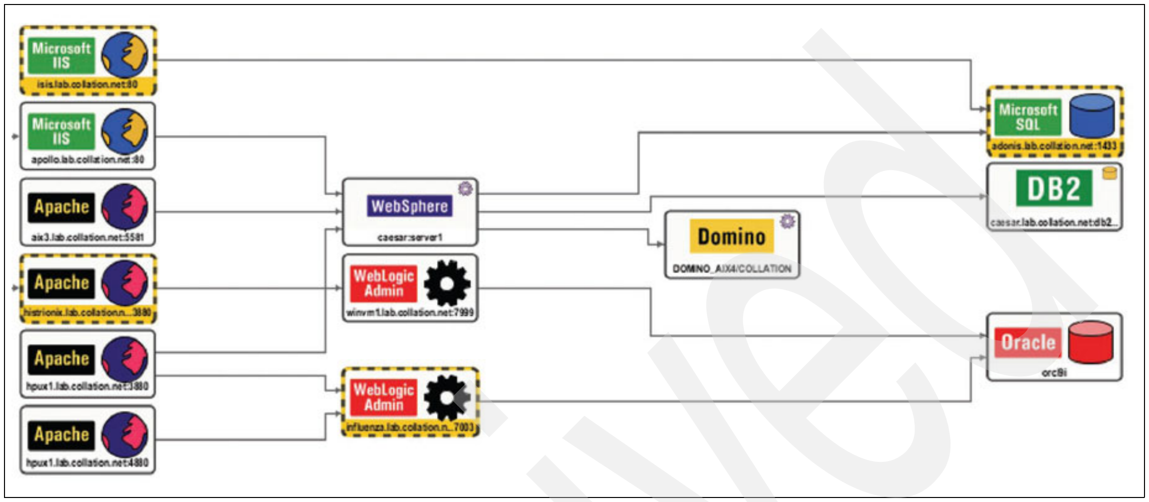

Microsoft IIS
isis.lab.collation.net:80
Microsoft IIS
apollo.lab.collation.net:80
Apache
aix3.lab.collation.net:5581
Apache
histrionix.lab.collation.n...3880
Apache
hpux1.lab.collation.net:3880
Apache
hpux1.lab.collation.net:4880
WebSphere
caesar:server1
WebLogic Admin
winvm1.lab.collation.net:7999
WebLogic Admin
influenza.lab.collation.n...7003
Domino
DOMINO_AIX4/COLLATION
Microsoft SQL
adonis.lab.collation.net:1433
DB2
caesar.lab.collation.net:db2...
Oracle
orc8i

Figure 8-1 Tivoli Change and Configuration Management Database

Tivoli Change and Configuration Management Database can automatically discover the following entities in an infrastructure environment:

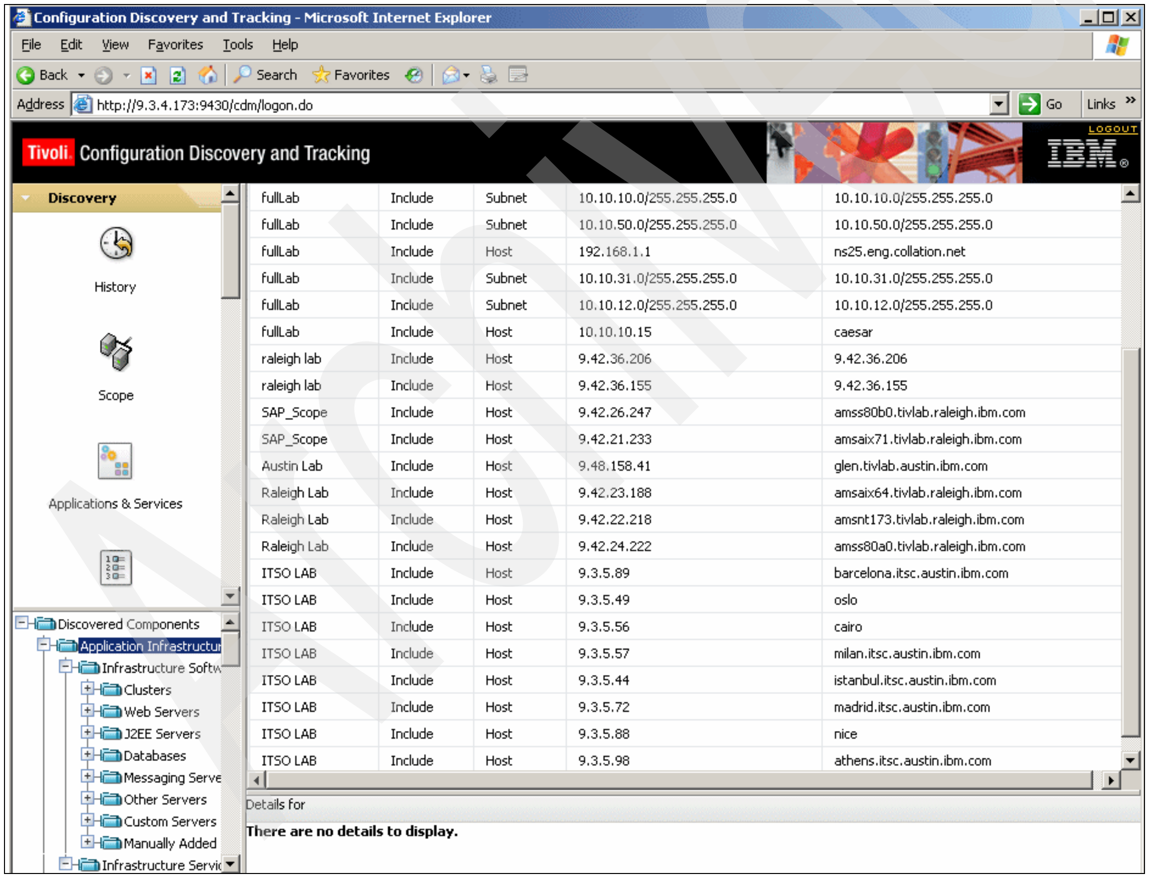
- Network
- System
- Application
- Infrastructure service components

Within Tivoli Dynamic Workload Broker you can define workstations in your environment as computers or logical resources. A computer is a workstation categorized on the basis of its hardware characteristics. A logical resource is a workstation that can be used to represent applications, groups, licenses, servers and any feature that may be applied to computer system or to other logical resources.

The integration with Tivoli Change and Configuration Management Database enhances the process of creating logical resources by importing resources stored in the Tivoli Change and Configuration Management Database.

Note: Tivoli Application Dependency Discovery (TADDM) can also be used in this scenario. Tivoli Application Dependency Discovery is the discovery engine of Tivoli Change and Configuration Management Database and is also available as a standalone product. For the purposes of Tivoli Dynamic Workload Broker, you can use either of the products.

Figure 8-2 shows the machines discovered by our Tivoli Change and Configuration Management Database installation. Note that barcelona, a Linux machine, is one of the machines discovered by the Tivoli Change and Configuration Management Database. We later use this information in our Tivoli Dynamic Workload Broker and Tivoli Change and Configuration Management Database scenario.



fullLab	Include	Subnet	10.10.10.0/255.255.255.0	10.10.10.0/255.255.255.0
fullLab	Include	Subnet	10.10.50.0/255.255.255.0	10.10.50.0/255.255.255.0
fullLab	Include	Host	192.168.1.1	ns25.eng.collation.net
fullLab	Include	Subnet	10.10.31.0/255.255.255.0	10.10.31.0/255.255.255.0
fullLab	Include	Subnet	10.10.12.0/255.255.255.0	10.10.12.0/255.255.255.0
fullLab	Include	Host	10.10.10.15	caesar
raleigh lab	Include	Host	9.42.36.206	9.42.36.206
raleigh lab	Include	Host	9.42.36.155	9.42.36.155
SAP_Scope	Include	Host	9.42.26.247	amss80b0.tivlab.raleigh.ibm.com
SAP_Scope	Include	Host	9.42.21.233	amsaix71.tivlab.raleigh.ibm.com
Austin Lab	Include	Host	9.48.158.41	glen.tivlab.austin.ibm.com
Raleigh Lab	Include	Host	9.42.23.188	amsaix64.tivlab.raleigh.ibm.com
Raleigh Lab	Include	Host	9.42.22.218	amsnt173.tivlab.raleigh.ibm.com
Raleigh Lab	Include	Host	9.42.24.222	amss80a0.tivlab.raleigh.ibm.com
ITSO LAB	Include	Host	9.3.5.89	barcelona.itsc.austin.ibm.com
ITSO LAB	Include	Host	9.3.5.49	oslo
ITSO LAB	Include	Host	9.3.5.56	cairo
ITSO LAB	Include	Host	9.3.5.57	milan.itsc.austin.ibm.com
ITSO LAB	Include	Host	9.3.5.44	istanbul.itsc.austin.ibm.com
ITSO LAB	Include	Host	9.3.5.72	madrid.itsc.austin.ibm.com
ITSO LAB	Include	Host	9.3.5.88	nice
ITSO LAB	Include	Host	9.3.5.98	athens.itsc.austin.ibm.com

Details for

There are no details to display.

Figure 8-2 Systems discovered by Tivoli Change and Configuration Management Database

8.2.1 Tivoli Change and Configuration Management configuration

Configuration information for the connection to the Tivoli Change and Configuration Management Database is stored in the TDWB_server_install_directory/config/CCMDBConfig.properties file. This file is created during the installation of the CCMDB enablement for Tivoli Dynamic Workload Broker.

- CCMDB.user

  Specifies the user ID to connect to Tivoli Change and Configuration Management Database.

- CCMDB.softwareElements

  A list of software elements to be imported. By default all software elements of the AppServer type will be processed. Only elements from the AppServer can be processed.

- CMDBAddress.port

  Port used to communicate with Tivoli Configuration Management Database. The default value is 9530.

- CCMDB.pwd

  Stored encrypted copy of user ID set in CCMDB.user. This value is filled in with an encrypted value after installation and if command line is invoked passing password.

- CCMDB.lastUpdate

  This value is updated with the time stamp of the last successful import. Anytime a full import is desired this needs to be reset to zero, otherwise only incremental changes from the datestamp are imported.

  Note: If a resource that has been imported is manually deleted from Tivoli Dynamic Workload Broker, the only way to reacquire this resource is to reset CCMDB.lastUpdate to zero and execute the **`ccmdbdataimport`** command.

- CMDBAddress.host

  The Tivoli Change and Configuration Management Database server address. This can be host name or TCP address.

8.2.2 Integration steps

Installing the enablement to support integration with Tivoli Change and Configuration Management Database can be performed after installation of the

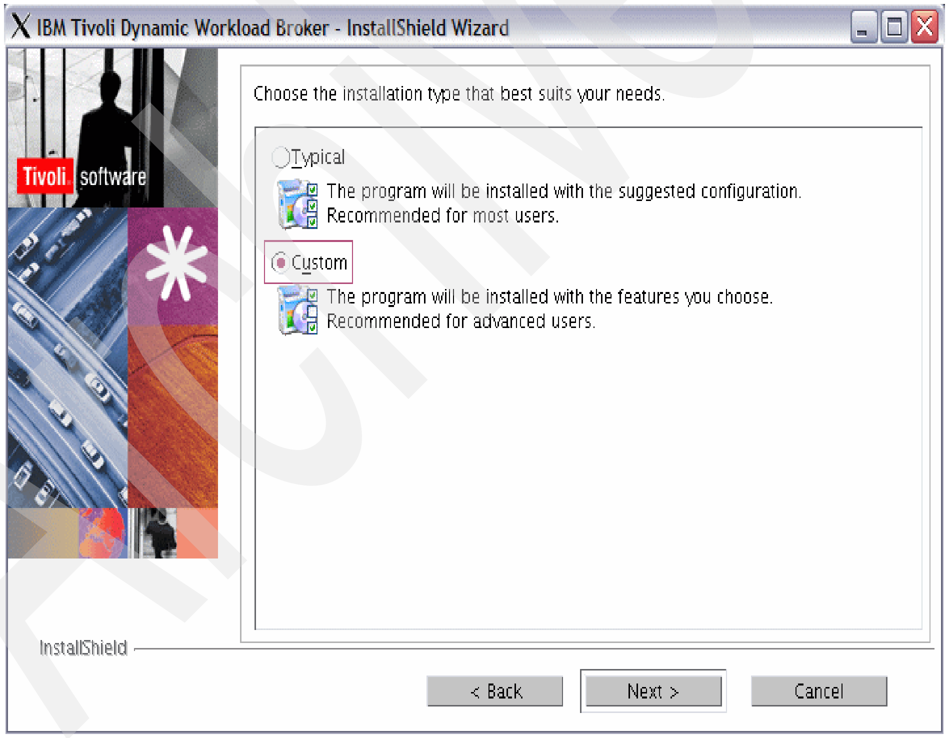
Tivoli Dynamic Workload Broker server, or can be installed with the initial installation of the Tivoli Dynamic Workload Broker server.

Once the Tivoli Change and Configuration Management Database enablement is installed, IBM Tivoli Dynamic Workload Broker is ready to import logical resource information from the Tivoli Change and Configuration Management Database. These logical resources are now available to be used for Tivoli Dynamic Workload Broker jobs.

Installing CCMDB enablement

To enable Tivoli Dynamic Workload Broker 1.1 for Tivoli Provisioning Manager enablement:

1. Insert CD 1, Tivoli Dynamic Workload Broker 1.1
2. Select the option to install the Tivoli Dynamic Workload Broker server.
3. Select the **Custom** option. See Figure 8-3.



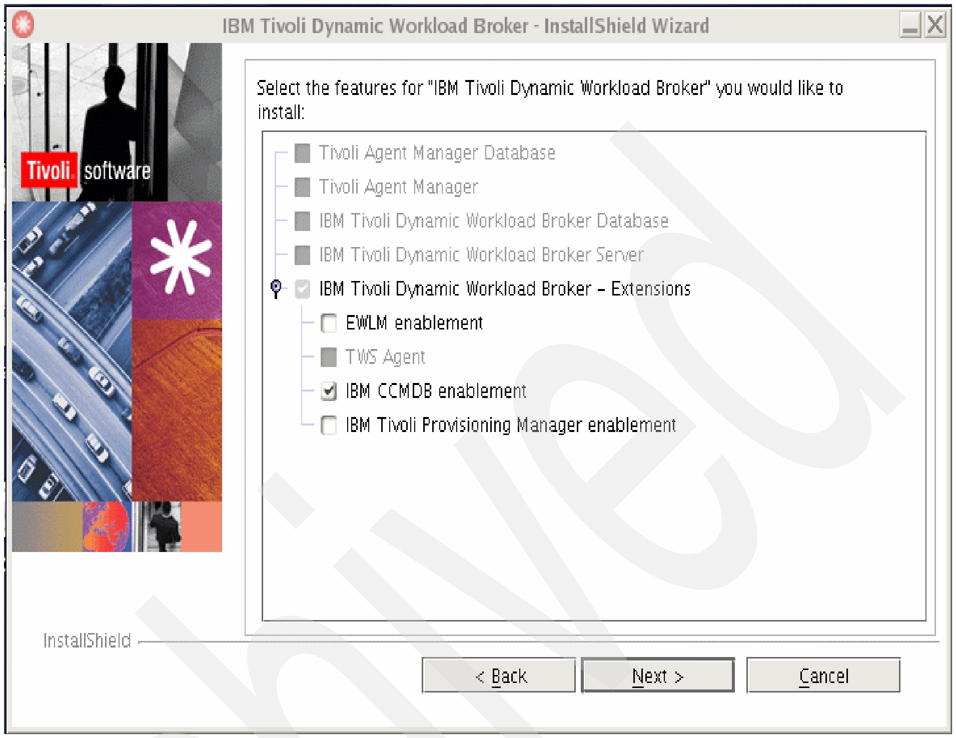
Choose the installation type that best suits your needs.

( ) Typical
The program will be installed with the suggested configuration. Recommended for most users.

(●) Custom
The program will be installed with the features you choose. Recommended for advanced users.

Figure 8-3 Install the Tivoli Change and Configuration Manager Database enablement

4. Select **IBM CCMDB enablement**, as shown in Figure 8-4 on page 307.

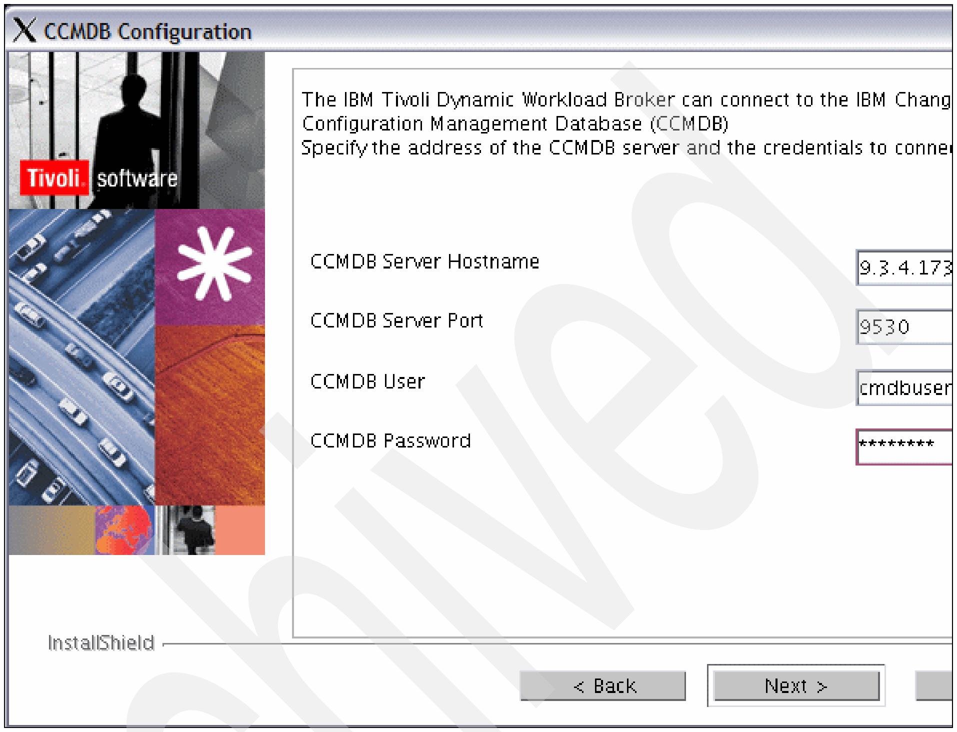

IBM Tivoli Dynamic Workload Broker - InstallShield Wizard

Select the features for "IBM Tivoli Dynamic Workload Broker" you would like to install:

- Tivoli Agent Manager Database
- Tivoli Agent Manager
- IBM Tivoli Dynamic Workload Broker Database
- IBM Tivoli Dynamic Workload Broker Server
- IBM Tivoli Dynamic Workload Broker – Extensions
  - [ ] EWLM enablement
  - TWS Agent
  - [✓] IBM CCMDB enablement
  - [ ] IBM Tivoli Provisioning Manager enablement

Figure 8-4 Install the Tivoli Change and Configuration Management Database

5. Provide the parameters for the Tivoli Change and Configuration Management Database, as shown in Figure 8-5. For a description of these parameters see "Tivoli Provisioning Manager configuration" on page 314.



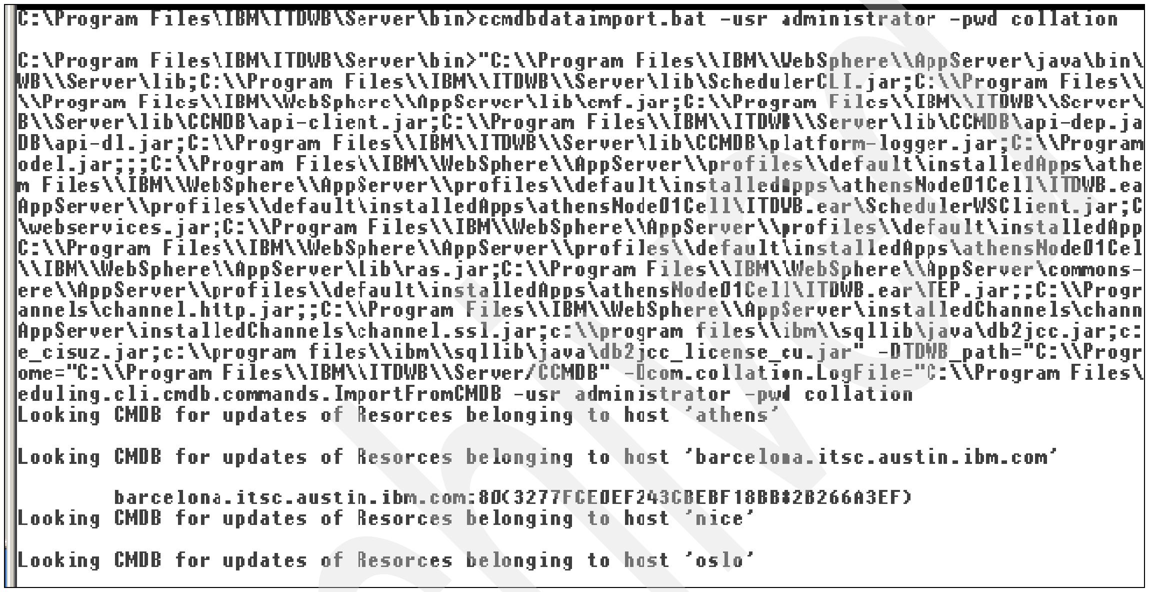
The IBM Tivoli Dynamic Workload Broker can connect to the IBM Chang
Configuration Management Database (CCMDB)
Specify the address of the CCMDB server and the credentials to conne

CCMDB Server Hostname	9.3.4.173
CCMDB Server Port	9530
CCMDB User	cmdbuser
CCMDB Password	*********

Figure 8-5 Tivoli Change and Configuration Management Database configuration.

Import resources from CCMDB

To initiate the import of resources, execute the **ccmdbdataimport** command, which is located in the installation_directory/bin. This command must be executed whenever changes to Tivoli Change and Configuration Database take place, in order to keep Tivoli Dynamic Workload Broker in sync. This command can be scheduled as a Tivoli Workload Scheduler daily job. Figure 8-6 shows the execution output of the **ccmdbdataimport** command.

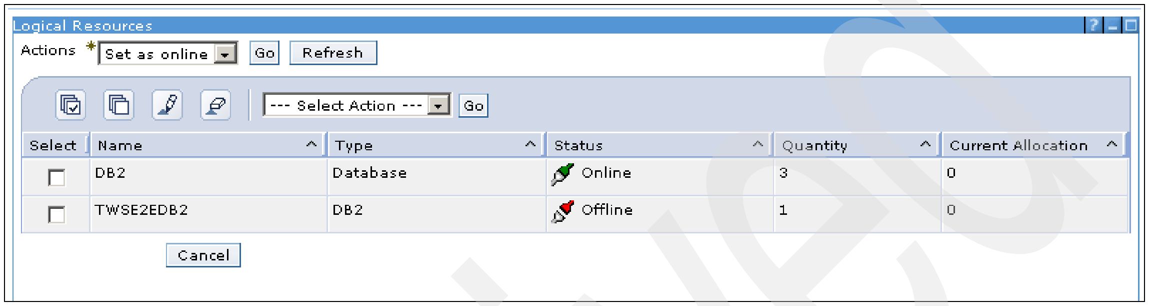


```
C:\Program Files\IBM\ITDWB\Server\bin>ccmdbdataimport.bat -usr administrator -pwd collation

C:\Program Files\IBM\ITDWB\Server\bin>"C:\\Program Files\\IBM\\WebSphere\\AppServer\java\bin\
WB\\Server\lib;C:\\Program Files\\IBM\\ITDWB\\Server\lib\SchedulerCLI.jar;C:\\Program Files\\
\\Program Files\\IBM\\WebSphere\\AppServer\lib\cmf.jar;C:\\Program Files\\IBM\\ITDWB\\Server\
B\\Server\lib\CCMDB\api-client.jar;C:\\Program Files\\IBM\\ITDWB\\Server\lib\CCMDB\api-dep.ja
DB\api-dl.jar;C:\\Program Files\\IBM\\ITDWB\\Server\lib\CCMDB\platform-logger.jar;C:\\Program
odel.jar;;;C:\\Program Files\\IBM\\WebSphere\\AppServer\\profiles\\default\installedApps\athe
m Files\\IBM\\WebSphere\\AppServer\\profiles\\default\installedApps\athensNode01Cell\ITDWB.ea
AppServer\\profiles\\default\installedApps\athensNode01Cell\ITDWB.ear\SchedulerWSClient.jar;C
\webservices.jar;C:\\Program Files\\IBM\\WebSphere\\AppServer\\profiles\\default\installedApp
C:\\Program Files\\IBM\\WebSphere\\AppServer\\profiles\\default\installedApps\athensNode01Cel
\\IBM\\WebSphere\\AppServer\lib\ras.jar;C:\\Program Files\\IBM\\WebSphere\\AppServer\commons-
ere\\AppServer\\profiles\\default\installedApps\athensNode01Cell\ITDWB.ear\TEP.jar;;C:\\Progr
annels\channel.http.jar;;C:\\Program Files\\IBM\\WebSphere\\AppServer\installedChannels\chann
AppServer\installedChannels\channel.ssl.jar;c:\\program files\\ibm\\sqllib\java\db2jcc.jar;c:
e_cisuz.jar;c:\\program files\\ibm\\sqllib\java\db2jcc_license_cu.jar" -DTDWB_path="C:\\Progr
ome="C:\\Program Files\\IBM\\ITDWB\\Server/CCMDB" -Dcom.collation.LogFile="C:\\Program Files\
eduling.cli.cmdb.commands.ImportFromCMDB -usr administrator -pwd collation
Looking CMDB for updates of Resources belonging to host 'athens'

Looking CMDB for updates of Resources belonging to host 'barcelona.itsc.austin.ibm.com'

        barcelona.itsc.austin.ibm.com:80(3277FCE0EF243CBEBF18BB#2B266A3EF)
Looking CMDB for updates of Resources belonging to host 'nice'

Looking CMDB for updates of Resources belonging to host 'oslo'
```

Figure 8-6 ccmdbdataimport execution output

In Figure 8-7 the output shows that only they systems that are executing the Tivoli Dynamic Workload Broker agent are checked for updates within the Tivoli Change and Configuration Management Database. At the time of execution only four systems had the Tivoli Dynamic Workload Broker agent executing: athens, barcelona, nice, and oslo.

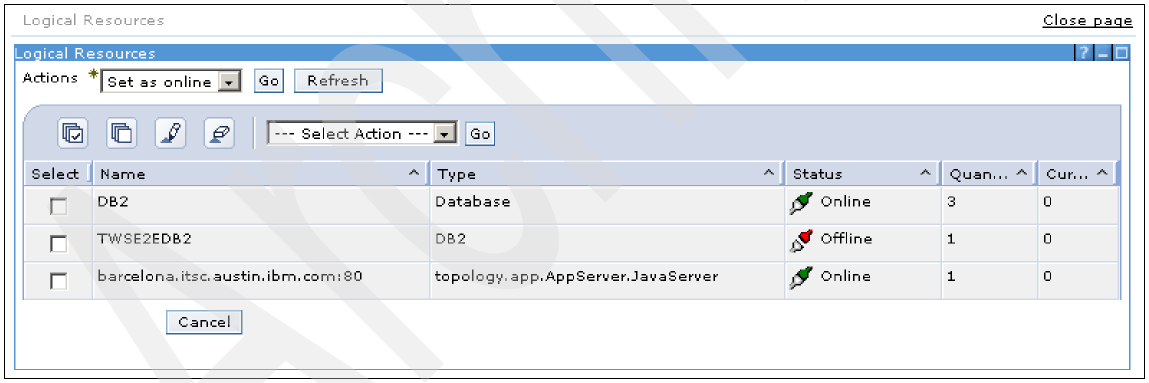
Previous to executing the **`ccmdbdataimport`** command, the Logical Resource screen shows the following (Figure 8-7).



Select	Name	Type	Status	Quantity	Current Allocation
☐	DB2	Database	Online	3	0
☐	TWSE2EDB2	DB2	Offline	1	0

Figure 8-7 Logical Resource screen before executing the ccmdbdataimport command

After executing the **`ccmdbdataimport`** command, the Logical Resource screen shows the following (Figure 8-8).



Select	Name	Type	Status	Quan...	Cur...
☐	DB2	Database	Online	3	0
☐	TWSE2EDB2	DB2	Offline	1	0
☐	barcelona.itsc.austin.ibm.com:80	topology.app.AppServer.JavaServer	Online	1	0

Figure 8-8 Logical Resource screen after executing the ccmdbdataimport command

A logical resource is created for each discovered Change and Configuration Management Database resource. A relationship of the Contains type is defined between the new logical resource and the IBM Tivoli Dynamic Workload Broker computer system defined in the RunsOn column in the Change and Configuration Management Database. See Table 8-2.

Table 8-2 Mapping between Tivoli Dynamic Workload Broker and Tivoli Change and Tivoli Change and Configuration Management Database attributes.

Tivoli Dynamic Workload Broker attribute	Change and Configuration Management Database attribute
Display name	Display name or Label if Display Name N/A
Name	GUID
Administrative status	Admin State
Change and Configuration Management Database subtype	Collation® Type
Creator name	CDMSource
Owner name	“CCMDB”
Quantity	1

Note: The resource matching is performed based on the fully qualified host name of the computer on which they run. Therefore the matching can be performed only for those resources running on Tivoli Dynamic Workload broker computer systems that have a fully qualified host name matching the host name listed in the RunsOn column in Tivoli Change and Configuration Management Database. Reviewing ccmdbdataimport output shows that only the system barcelona was fully qualified and thus the only system to import a logical resource.

Review the integration flow depicted in Figure 8-9.

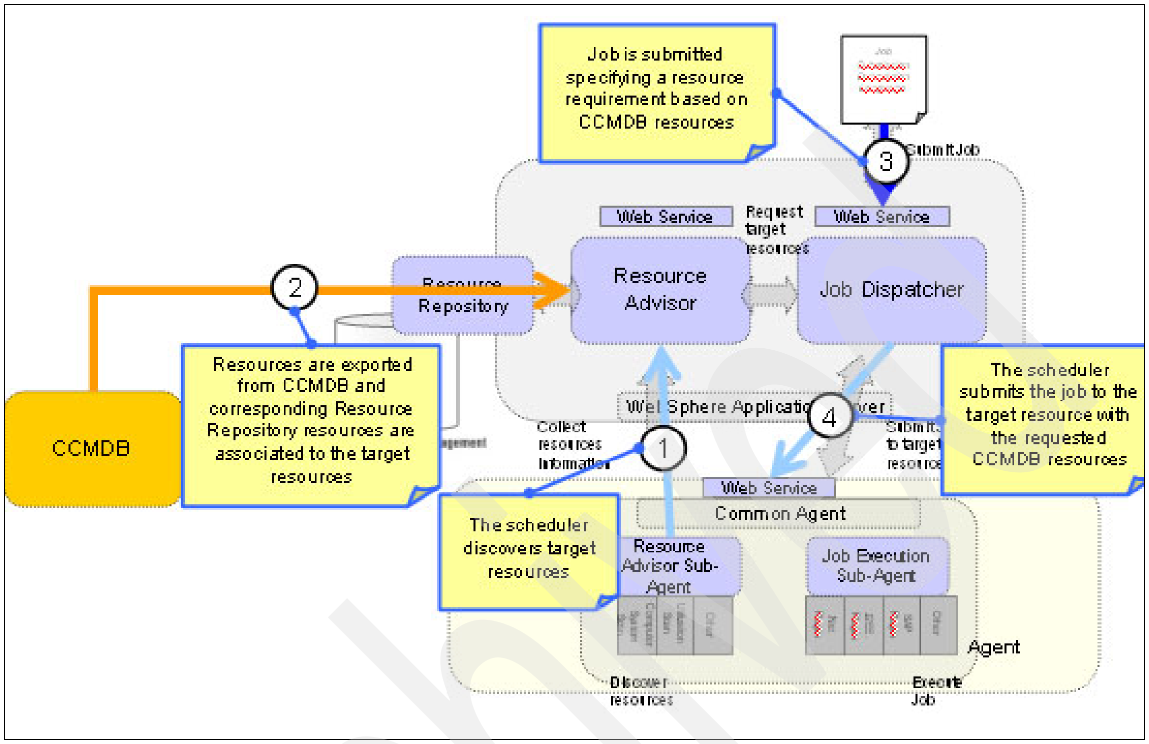

Job is submitted specifying a resource requirement based on CCMDB resources
SubmitJob
3
Web Service
Request target resources
Web Service
Resource Repository
2
Resource Advisor
Job Dispatcher
Resources are exported from CCMDB and corresponding Resource Repository resources are associated to the target resources
CCMDB
WebSphere Application Server
4
Collect resources Information
1
Submit to target resource
The scheduler submits the job to the target resource with the requested CCMDB resources
Web Service
Common Agent
The scheduler discovers target resources
Resource Advisor Sub-Agent
Job Execution Sub-Agent
Agent
Discover resources
Execute Job

Figure 8-9 Tivoli Dynamic Workload Broker and Tivoli Change and Configuration Management Database integration architecture

The flow is:

1. Tivoli Dynamic Workload Broker discovers target resources.
2. Resources are imported from the Tivoli Change and Configuration Management Database and corresponding Resource Repository resources are associated to the target resource.
3. The job is submitted specifying a resource requirement based on Tivoli Change and Configuration Management Database resources.
4. The Tivoli Dynamic Workload Broker submits the job to the target resource with the requested Tivoli Change and Configuration Management Database resources.

For troubleshooting the Tivoli Change and Configuration Management Database integration refer to 10.6, “Troubleshooting the integration with CCMDB” on page 513.

8.3 Integration with IBM Tivoli Provisioning Manager

IBM Tivoli Provisioning Manager (Tivoli Provisioning Manager) is an automated resource management solution for corporate and Internet data centers. Through orchestrated provisioning, it provides the ability to manage the IT environment in real time, according to defined business policies, to achieve desired business goals.

Integrating Tivoli Dynamic Workload Broker with Tivoli Provisioning Manager provides the capability of launching Tivoli Provisioning Manager workflows as part of recovery actions to be performed when the resources required by a job are not available. Using the Job Brokering Definition Console, specify the name of a Tivoli Provisioning Manager workflow to be run as the recovery action in the Scheduling pane. The recovery action is automatically started when a job submission times out due to unavailable resources.

Figure 8-10 shows the architecture of Tivoli Provisioning Manager integration.

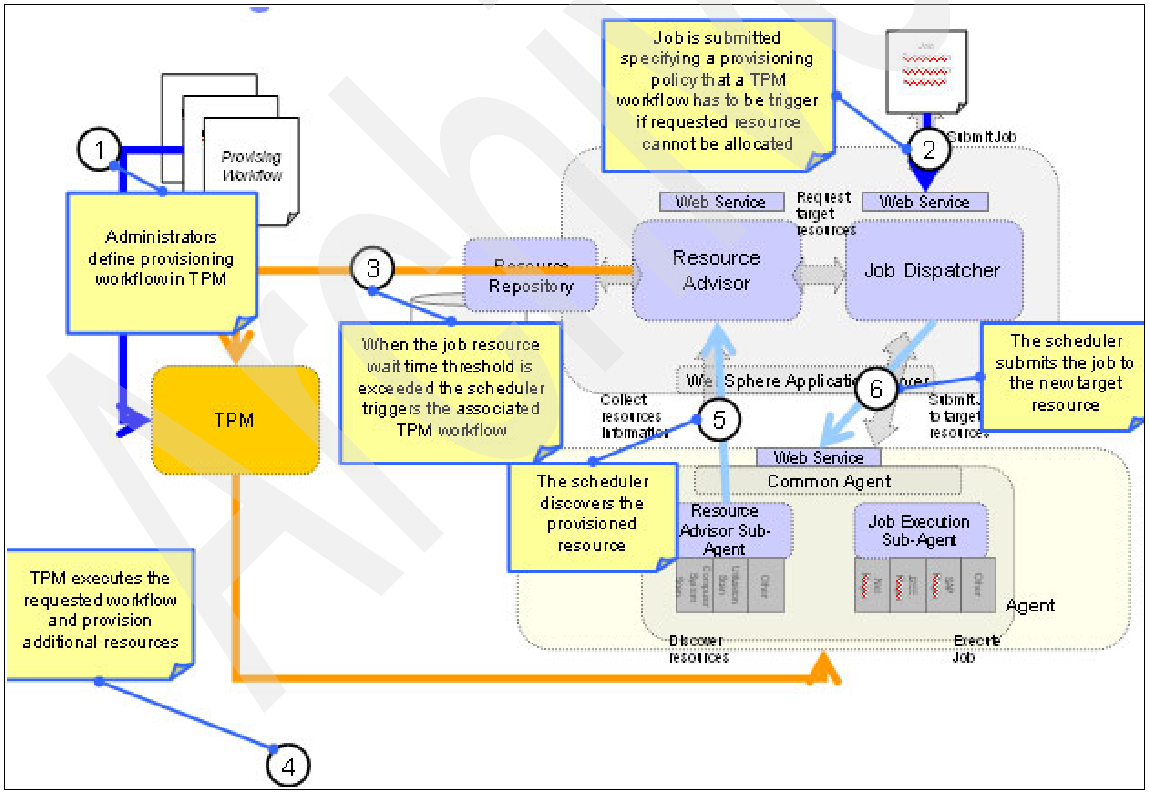

1
Provising Workflow
Administrators define provisioning workflow in TPM
TPM
TPM executes the requested workflow and provision additional resources
4
3
Resource Repository
When the job resource wait time threshold is exceeded the scheduler triggers the associated TPM workflow
Job is submitted specifying a provisioning policy that a TPM workflow has to be trigger if requested resource cannot be allocated
2
Submit Job
Web Service
Request target resources
Web Service
Resource Advisor
Job Dispatcher
WebSphere Application Server
5
6
Collect resources information
The scheduler discovers the provisioned resource
The scheduler submits the job to the new target resource
Submit to target resources
Web Service
Common Agent
Resource Advisor Sub-Agent
Job Execution Sub-Agent
Agent
Discover resources
Execute Job

Figure 8-10 Tivoli Provisioning Manager integration

Note: For more information about Tivoli Provisioning Manager refer to *Deployment Guide Series: IBM Tivoli Provisioning Manager Version 5.1,* SG24-7261.

8.3.1 Tivoli Provisioning Manager configuration

Configuration information for the connection to the Tivoli Provisioning Manager server is stored in the TDWB_server_install_directory/config/TPMConfig.properties file. This file is created during the installation of the Tivoli Provisioning Manager enablement for the Tivoli Dynamic Workload Broker.

Parameters in this file can be overridden in a single job when creating the job with the Job Brokering Definition Console.

The following parameters are available in the TPMConfig.properties file:

- TPMAddress.hostname

  Specifies the host name of the Tivoli Provisioning Manager server to be used when running the recovery action. Use a fully qualified host name.

- TPMAddress.port

  Specifies the port number of the Tivoli Provisioning Manager server to be used when running the recovery action. The Tivoli Provisioning Manager administrator must supply this information. The default is 8777.

- TPM.user

  Specifies the Tivoli Provisioning Manager user name of a user with the authority to run workflows. The default is tioappadmin.

- TPM.pwd

  Specifies the password for the Tivoli Provisioning Manager user to be used when running a Tivoli Provisioning Manager workflow.

8.3.2 Integration steps

Installing the enablement to support integration with Tivoli Provisioning Manager can be performed post install of the Tivoli Dynamic Workload Broker server, or can be installed with the initial installation of the Tivoli Dynamic Workload Broker server.

Once the Tivoli Provisioning Manager enablement is installed IBM Tivoli Dynamic Workload Broker is ready to execute Tivoli Provisioning Manager workflows.

The following steps are required for Tivoli Dynamic Workload Broker jobs to execute Tivoli Provisioning Manager Workflows.

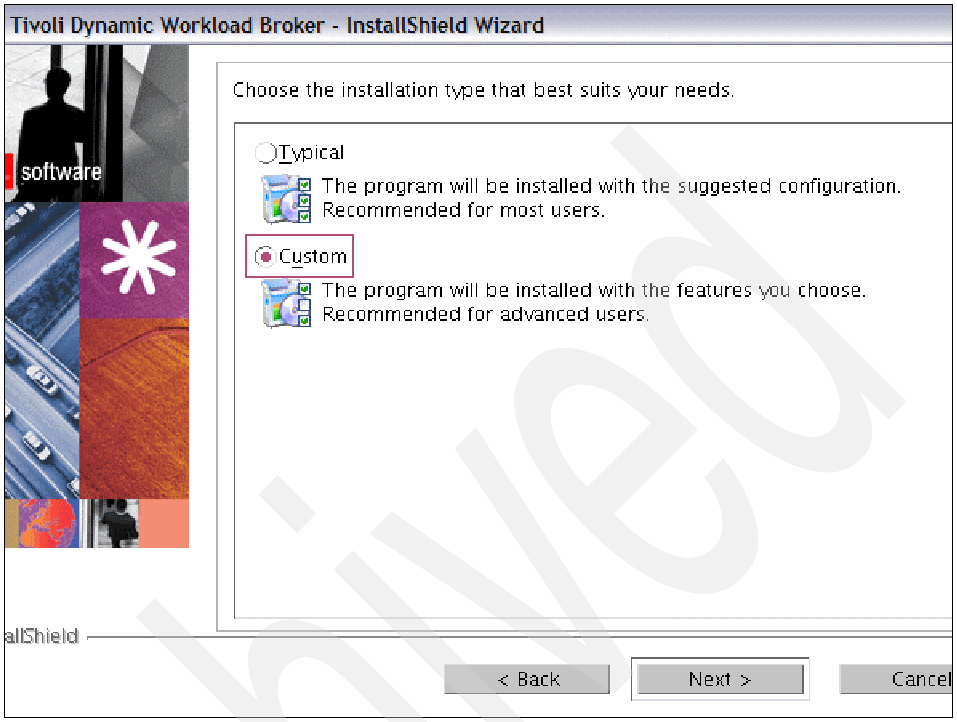
1. Install Tivoli Provisioning Manager enablement on the IBM Tivoli Dynamic Workload Broker server. This is on the IBM Tivoli Dynamic Workload Broker CD (CD_1).
2. Define a Tivoli Provisioning Manager Workflow. This step is usually performed by the Tivoli Provisioning Manager administrator. In our example the Tivoli Provisioning Manager workflow installs the IBM Tivoli Dynamic Workload Broker agent on a Linux system.
3. Create or modify an existing IBM Tivoli Dynamic Workload Broker job definition using the Job Broker Definition Console, which specifies the Tivoli Provisioning Manager workflow as the recovery action for the job. This is done under the Scheduling pane of the job definition.

Install the Tivoli Provisioning Manager enablement

Do the following to enable Tivoli Dynamic Workload Broker 1.1 for Tivoli Provisioning Manager enablement.

1. Insert CD 1, Tivoli Dynamic Workload Broker 1.1.
2. Select the option to install the Tivoli Dynamic Workload Broker server.

3. Select **Custom**. See Figure 8-11.



Tivoli Dynamic Workload Broker - InstallShield Wizard

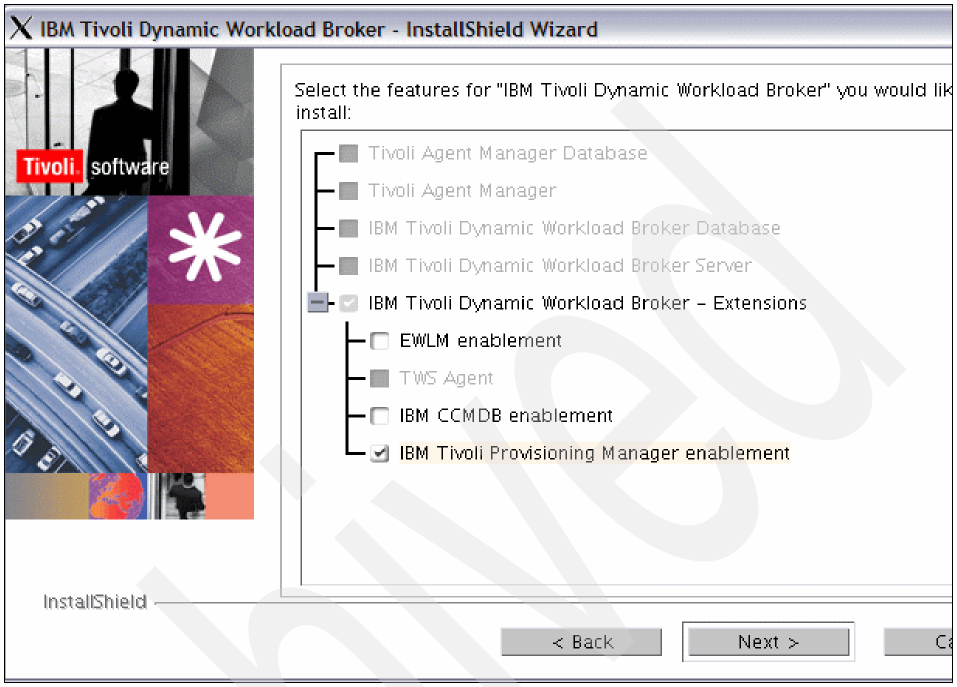
Choose the installation type that best suits your needs.

○ Typical
The program will be installed with the suggested configuration.
Recommended for most users.

● Custom
The program will be installed with the features you choose.
Recommended for advanced users.

Figure 8-11 Install the Tivoli Provisioning Manager workload enablement

4. Select **IBM Tivoli Provisioning Manager enablement,** as shown in Figure 8-12.

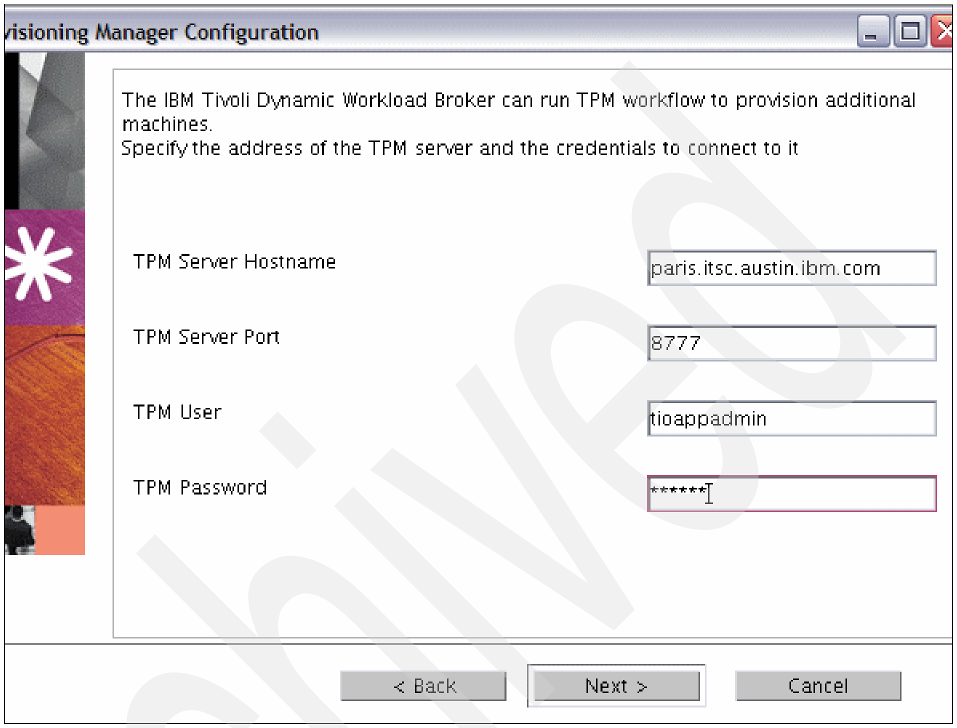


Select the features for "IBM Tivoli Dynamic Workload Broker" you would lik install:

- Tivoli Agent Manager Database
- Tivoli Agent Manager
- IBM Tivoli Dynamic Workload Broker Database
- IBM Tivoli Dynamic Workload Broker Server
- IBM Tivoli Dynamic Workload Broker - Extensions
  - [ ] EWLM enablement
  - TWS Agent
  - [ ] IBM CCMDB enablement
  - [✓] IBM Tivoli Provisioning Manager enablement

Figure 8-12 Install the Tivoli Provisioning Manager workload enablement

5. Provide the parameters for IBM Tivoli Provisioning Manager enablement, as shown in Figure 8-13. For a description of these parameters, see “Tivoli Provisioning Manager configuration” on page 314.



The IBM Tivoli Dynamic Workload Broker can run TPM workflow to provision additional machines.
Specify the address of the TPM server and the credentials to connect to it

TPM Server Hostname	paris.itsc.austin.ibm.com
TPM Server Port	8777
TPM User	tioappadmin
TPM Password	******

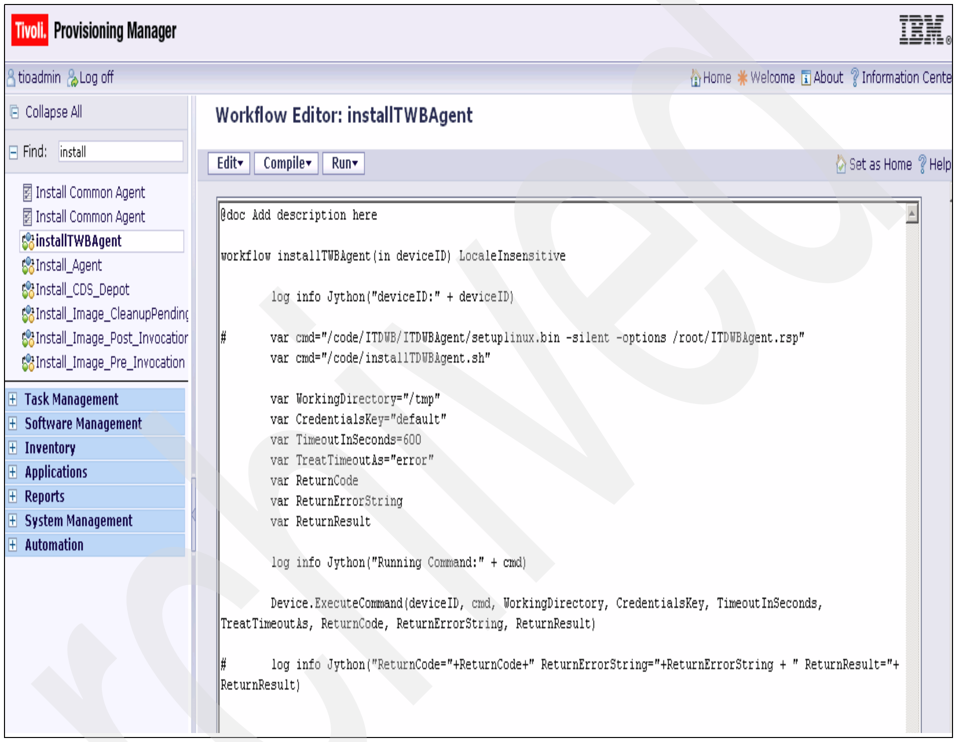
Figure 8-13 IBM Tivoli Provisioning Manager configuration

Create a Tivoli Provisioning Manager workflow

The next step is to develop a workflow in Tivoli Provisioning Manager that will provision the resources required by Tivoli Workload Broker jobs. A workflow in Tivoli Provisioning Manager is a simple program with a number of constructs that is used to manage an environment. In this example the workflow simply installs a Tivoli Dynamic Workload agent on a Linux system, *oslo*.

If you are not familiar with creating Tivoli Provisioning Manager workflows, you can refer to *Developing Workflows and Automation Packages for IBM Tivoli Intelligent Orchestrator V3.1*, SG24-6057.

Note: Tivoli Intelligent Orchestrator is a product that automatically triggers the provisioning, configuration, and deployment performed by Tivoli Provisioning Manager, which is part of the Tivoli Intelligent Orchestrator product, as well as being a standalone product. So the discussion here regarding the Tivoli Provisioning Manager also applies to Tivoli Intelligent Orchestrator.

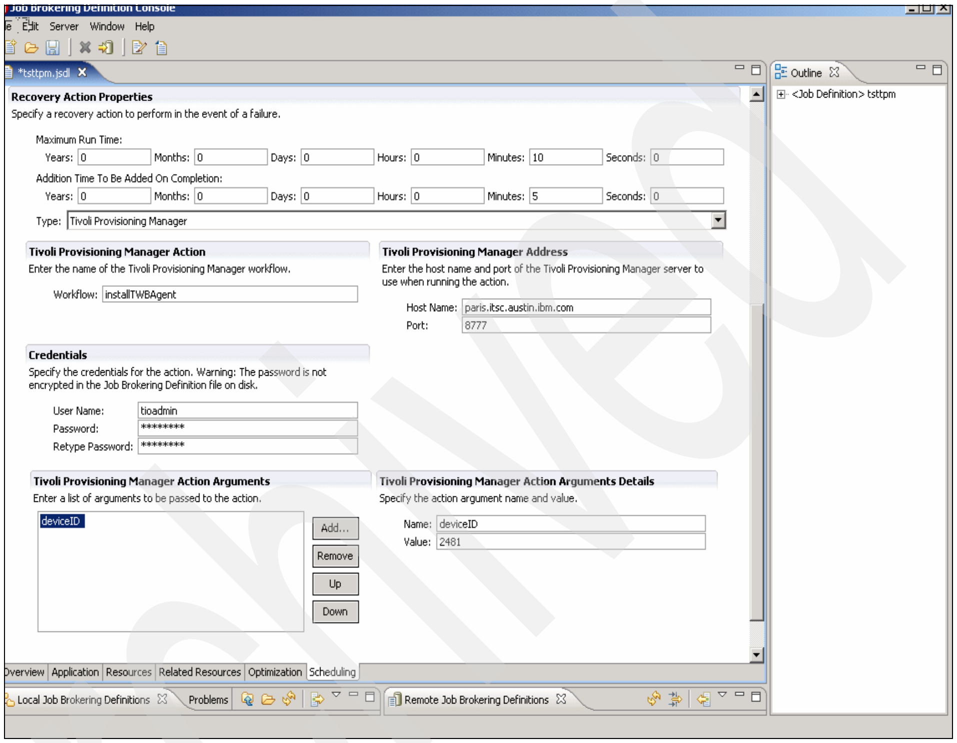


```
@doc Add description here

workflow installTWBAgent(in deviceID) LocaleInsensitive

        log info Jython("deviceID:" + deviceID)

#       var cmd="/code/ITDWB/ITDWBAgent/setuplinux.bin -silent -options /root/ITDWBAgent.rsp"
        var cmd="/code/installITDWBAgent.sh"

        var WorkingDirectory="/tmp"
        var CredentialsKey="default"
        var TimeoutInSeconds=600
        var TreatTimeoutAs="error"
        var ReturnCode
        var ReturnErrorString
        var ReturnResult

        log info Jython("Running Command:" + cmd)

        Device.ExecuteCommand(deviceID, cmd, WorkingDirectory, CredentialsKey, TimeoutInSeconds,
TreatTimeoutAs, ReturnCode, ReturnErrorString, ReturnResult)

#       log info Jython("ReturnCode="+ReturnCode+" ReturnErrorString="+ReturnErrorString + " ReturnResult="+
ReturnResult)
```

Figure 8-14 Tivoli Provisioning Manager workflow

Create IBM Tivoli Dynamic Workload Broker job definition

In our example we have created a simple Tivoli Workload Broker job definition that must be executed on a Linux system. Figure 8-15 shows the Tivoli Dynamic Workload Broker job that was created. The workflow is defined under the Scheduling tab.



Recovery Action Properties

Specify a recovery action to perform in the event of a failure.

Maximum Run Time:

Years: 0 Months: 0 Days: 0 Hours: 0 Minutes: 10 Seconds: 0

Addition Time To Be Added On Completion:

Years: 0 Months: 0 Days: 0 Hours: 0 Minutes: 5 Seconds: 0

Type: Tivoli Provisioning Manager

Tivoli Provisioning Manager Action

Enter the name of the Tivoli Provisioning Manager workflow.

Workflow: installTWBAgent

Tivoli Provisioning Manager Address

Enter the host name and port of the Tivoli Provisioning Manager server to use when running the action.

Host Name: paris.itsc.austin.ibm.com

Port: 8777

Credentials

Specify the credentials for the action. Warning: The password is not encrypted in the Job Brokering Definition file on disk.

User Name: tioadmin

Password: ********

Retype Password: ********

Tivoli Provisioning Manager Action Arguments

Enter a list of arguments to be passed to the action.

deviceID

Tivoli Provisioning Manager Action Arguments Details

Specify the action argument name and value.

Name: deviceID

Value: 2481

Figure 8-15 Tivoli Workload Broker job definition

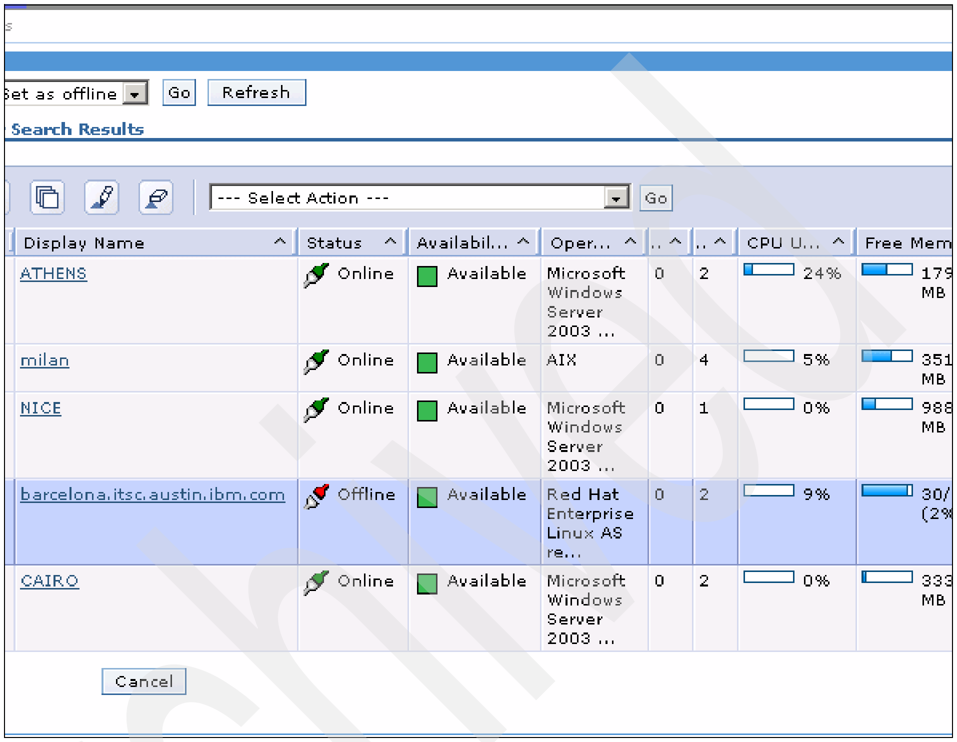
Example 8-1 .jsdl definition for this job

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
xmlns:jsdltpm="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdltpm"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdltp
m TPMAction.xsd
http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl JSDL.xsd
http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle
```

```
JSDL-Native.xsd" description="dirlinux" name="tsttpm">
  <jsdl:application name="executable">
    <jsdle:executable>
      <jsdle:arguments>
        <jsdle:value>-l</jsdle:value>
        <jsdle:value>/</jsdle:value>
      </jsdle:arguments>
      <jsdle:script>/bin/ls</jsdle:script>
    </jsdle:executable>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="LINUX"/>
    </jsdl:candidateOperatingSystems>
  </jsdl:resources>
  <jsdl:scheduling>
    <jsdl:recoveryActions>
      <jsdl:action additionalTimeOnCompletion="P0Y0M0DT0H5M0S"
maximumExecutionTime="P0Y0M0DT0H10M0S"
name="tpmaction">
        <jsdltpm:tpmaction workFlow="installTWBAgent">
          <jsdltpm:parameters>
            <jsdltpm:parameter name="deviceID">2481</jsdltpm:parameter>
          </jsdltpm:parameters>
          <jsdltpm:credential>
            <jsdl:userName>tioadmin</jsdl:userName>
            <jsdl:password>object00</jsdl:password>
          </jsdltpm:credential>
          <jsdltpm:tpmaddress host="paris.itsc.austin.ibm.com"
port="8777"/>
        </jsdltpm:tpmaction>
      </jsdl:action>
    </jsdl:recoveryActions>
<jsdl:maximumResourceWaitingTime>P0Y0M0DT0H0S</jsdl:maximumResourceWait
ingTime>
    <jsdl:estimatedDuration>P0Y0M0DT0H0S</jsdl:estimatedDuration>
    <jsdl:priority>0</jsdl:priority>
  </jsdl:scheduling>
</jsdl:jobDefinition>
```

Take the existing Linux Agent offline

We test the Tivoli Provisioning Manager integration by first taking the existing Linux Agent offline, as shown in Figure 8-16.

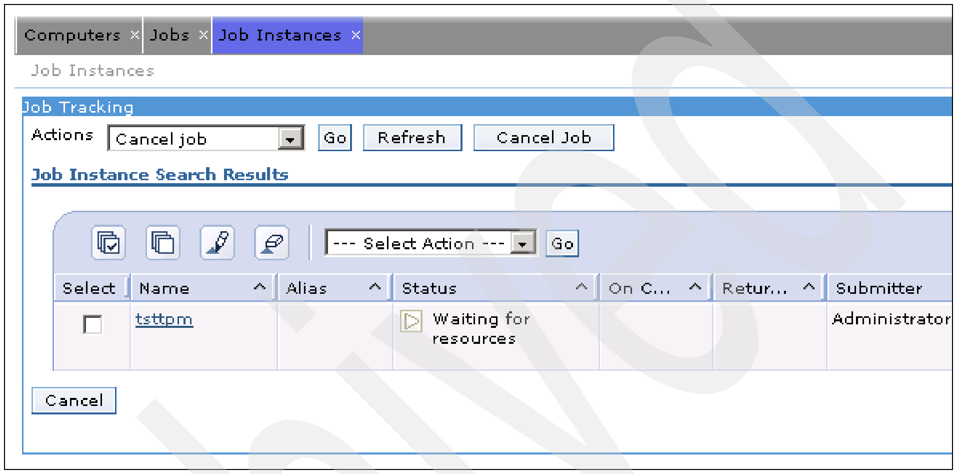


Display Name	Status	Availabil...	Oper...	..	..	CPU U...	Free Mem
ATHENS	Online	🟩 Available	Microsoft Windows Server 2003 ...	0	2	24%	179 MB
milan	Online	🟩 Available	AIX	0	4	5%	351 MB
NICE	Online	🟩 Available	Microsoft Windows Server 2003 ...	0	1	0%	988 MB
barcelona.itsc.austin.ibm.com	Offline	🟩 Available	Red Hat Enterprise Linux AS re...	0	2	9%	30/ (2%
CAIRO	Online	🟩 Available	Microsoft Windows Server 2003 ...	0	2	0%	333 MB

Figure 8-16 Take the existing Linux Agent offline

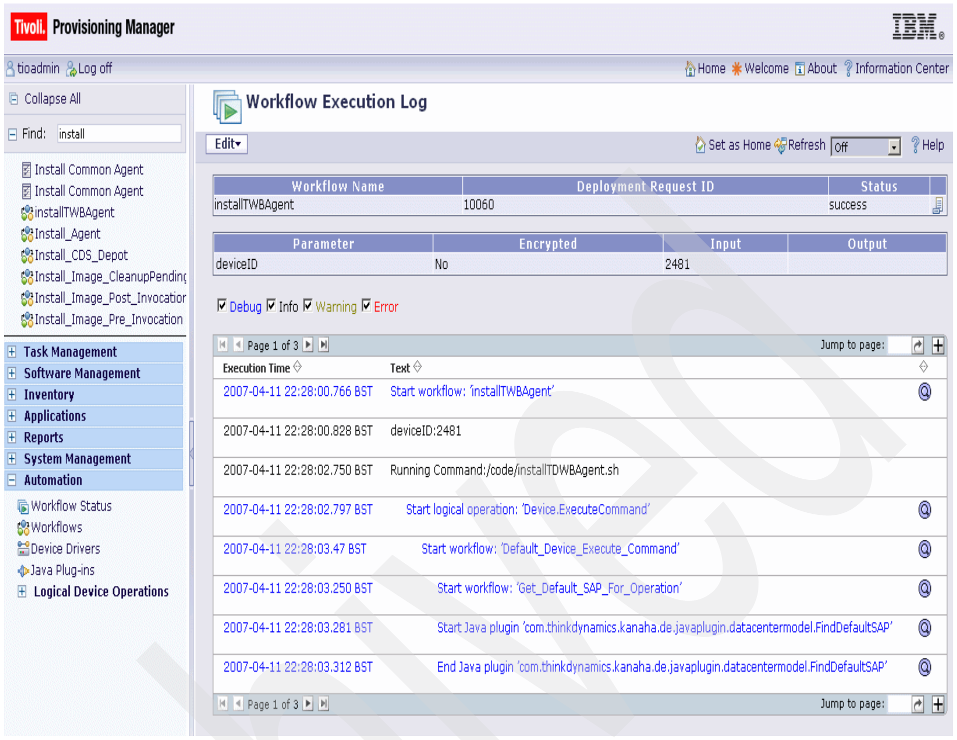
Submit the test job that requires provisioning of new resource

Note that the job will be waiting for resources, as shown in Figure 8-17. After the time out is reached for the job, the recovery action is initiated, and Tivoli Provisioning Manager executes its workflow, as shown in Figure 8-18 on page 324. Once the Tivoli Provisioning Manager workflow has executed, the job runs on the newly added Linux system, *oslo,* as shown in Figure 8-19 on page 324.



Select	Name	Alias	Status	On C...	Retur...	Submitter
☐	tsttpm		Waiting for resources			Administrator

Figure 8-17 Job waiting for resources

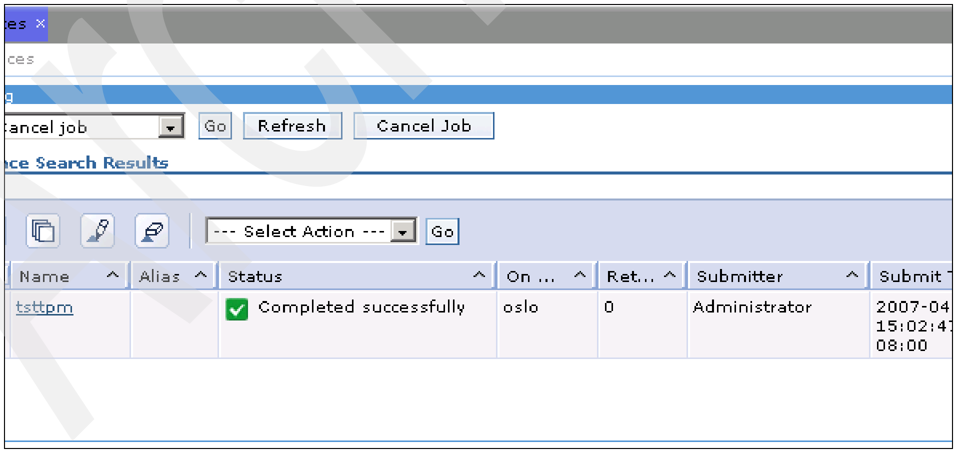


Workflow Name	Deployment Request ID	Status
installTWBAgent	10060	success

Parameter	Encrypted	Input	Output
deviceID	No	2481	

```
Execution Time              Text
2007-04-11 22:28:00.766 BST Start workflow: 'installTWBAgent'
2007-04-11 22:28:00.828 BST deviceID:2481
2007-04-11 22:28:02.750 BST Running Command:/code/installTDWBAgent.sh
2007-04-11 22:28:02.797 BST   Start logical operation: 'Device.ExecuteCommand'
2007-04-11 22:28:03.47 BST      Start workflow: 'Default_Device_Execute_Command'
2007-04-11 22:28:03.250 BST       Start workflow: 'Get_Default_SAP_For_Operation'
2007-04-11 22:28:03.281 BST       Start Java plugin 'com.thinkdynamics.kanaha.de.javaplugin.datacentermodel.FindDefaultSAP'
2007-04-11 22:28:03.312 BST       End Java plugin 'com.thinkdynamics.kanaha.de.javaplugin.datacentermodel.FindDefaultSAP'
```

Figure 8-18 Tivoli Provisioning Manager workflow execution log



Name	Alias	Status	On ...	Ret...	Submitter	Submit
tsttpm		✓ Completed successfully	oslo	0	Administrator	2007-04 15:02:4 08:00

Figure 8-19 Job runs successfully on oslo

8.4 Integration with IBM Tivoli Monitoring

Monitoring the enterprise is the key point for meeting strict SLA requirements. Availability of servers, databases, and various applications, their response times, various logs, and other properties are analyzed and correlated in centralized monitoring solutions.

When a scheduling environment is in place, it is important for the system operator to know what the status of the monitoring environment is and how successful the jobs are that are being run within that environment. Did they run okay? Did some jobs not finish in time? Have some of the jobs not started yet because of lack of suitable free resources?

These are the important questions that can be answered only by implementing a sophisticated monitoring solution.

The central point of IBM Tivoli Monitoring (Tivoli Monitoring) portfolio is Tivoli Monitoring with its presentation layer, Tivoli Enterprise Portal (TEP). This section describes how Tivoli Dynamic Workload Broker and Tivoli Monitoring can be integrated together, so that important events occurring within Tivoli Dynamic Workload Broker can appear in Tivoli Enterprise Portal.

8.4.1 Tivoli Monitoring components and terminology

In this section we describe the Tivoli Monitoring (ITM) components. We also explain the basic terms necessary for understanding the Tivoli Monitoring concepts.

The Tivoli Enterprise Monitoring Server (TEMS)

The Tivoli Enterprise Monitoring Server is the central repository of data that comes from the Tivoli Enterprise Monitoring Agents. TEMS stores the definitions for conditions that indicate possible problems with monitored resources.

Tivoli Enterprise Portal Server (TEPS)

The Tivoli Enterprise Portal Server functions as a repository for all user data, such as the user IDs and user access control for the monitoring data, meaning what data each user will be able to access and how it is displayed.

Tivoli Enterprise Portal (TEP)

The Tivoli Enterprise Portal is the presentation layer for displaying the monitoring data and offers a consolidated view of the entire IT environment. Tivoli Enterprise Portal is a client application that connects to the Tivoli Enterprise Portal Server. See the example in Figure 8-20.

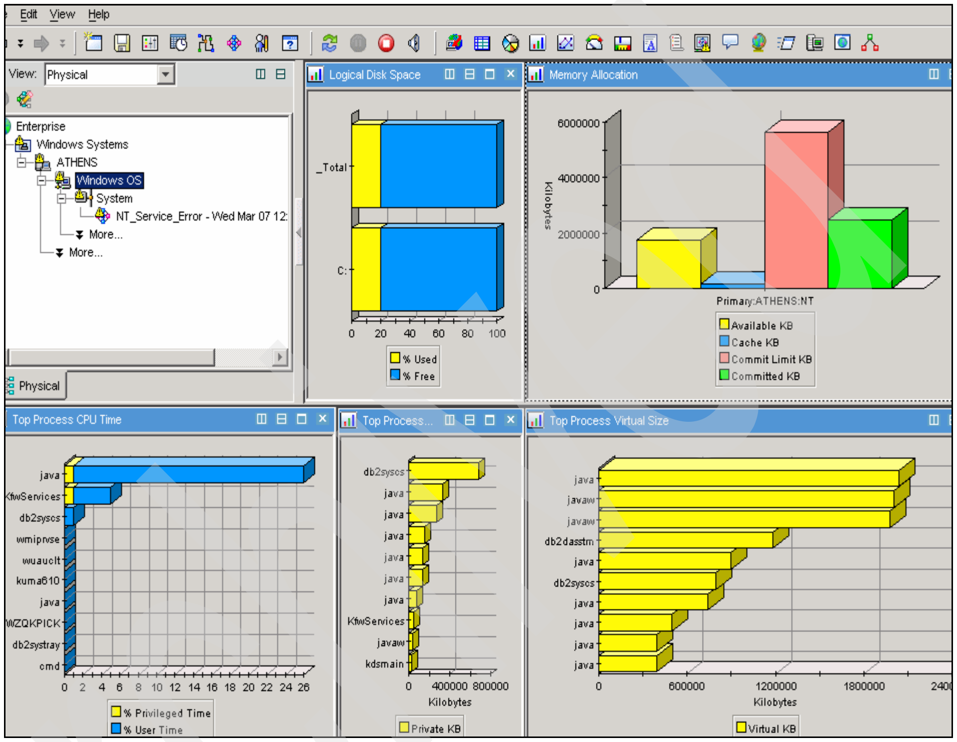


Figure 8-20 Tivoli Enterprise Portal interface

Tivoli Enterprise Monitoring Agent

There are several types of Tivoli Enterprise Monitoring Agents. They are distinguished by the purpose that they were designed for:

- The *Operating System Agent* is an agent that resides on the monitored server. It checks predefined values (for example, free space on disks, memory usage, and so on) and sends these values to the Tivoli Enterprise Monitoring Server (TEMS). Operating systems agents are provided out-of-the-box with the Tivoli Monitoring distribution.
- The *Application Agent* is an agent that is dedicated to monitor specific applications, such as Active directory, Lotus Domino® infrastructure, and so on. Each of these agents is specially programmed for its single-usage

purpose. It uses its API of monitored applications for gathering the necessary information.

- The *Database Agent* is similar to the article above. This is used for monitoring various RDBMS engines, such as DB2, Oracle, MS SQL, and so on.
- The *Universal Agent (UA)* is the only multi-purpose programmable agent. The universal agent can be configured to read data from various input sources, such as SNMP traps, incoming TCP traffic on specified ports, text log files, and so on. The universal agent is set up via configuration contained in metafiles. Those metafiles carry information that specifys how the monitored entities should be parsed. For a text file it is a pattern definition that specifys what lines in the log files are important, and thus are sent to the Tivoli Monitoring server. On the server side that information is displayed in the Tivoli Enterprise Portal. A *situation* can then be defined to reflect the content of incoming information. The term *situation* is explained next.

Figure 8-21 shows how the components of Tivoli Monitoring work together.

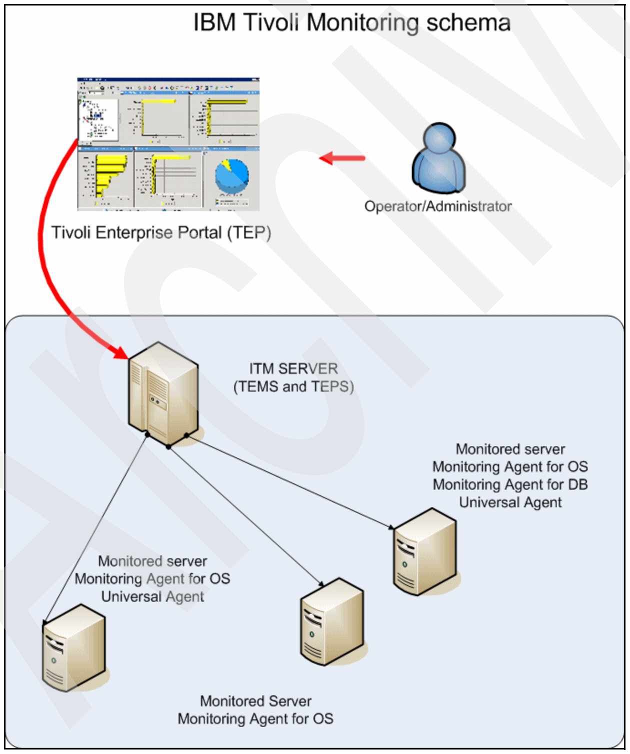


Figure 8-21 Tivoli Monitoring schema

Situation

Simply said, a *situation* is something interesting that occurred on one or more monitored entities. A typical example can be the situation in which free space on a logical volume decreases below a defined threshold (for instance, less than 5%). Situations can be of various *severities*, depending on their importance. For instance, a *critical* situation can be defined for cases when important servers are down. On the other hand, a *warning* situation can describe just the state in which one of the network adapters gets overloaded within the specified period of time.

Tivoli Monitoring comes with many out-of-box predefined situations. New situations can be defined using the Tivoli Enterprise Portal.

Note: Those who are familiar with other monitoring solutions, such as Tivoli Enterprise Console (TEC), may know the term *event*. The difference between an *event* and a *situation* is based on the fact that any data that flows to the Tivoli Enterprise Console are called *events*. Data that flows to the Tivoli Monitoring server are not *situations or events*. You can define a situation and when the conditions of the situation are met, then an event is generated. Both terms *event* and *situation* represent something *interesting* (in most cases something bad) that occurred in the IT environment. Thus, from this perspective, the term *event* in the Tivoli Enterprise Console and the term *situation* in Tivoli Monitoring are similar.

Automatic corrective action

If a threshold is exceeded and a situation is generated, it signals in most cases that something bad has occurred in the IT environment. As a response to this, a *corrective action* can be fired. A corrective action can be a script or predefined action that can be launched either on the Tivoli Monitoring server or on the monitored server (via monitoring agent). A typical example of a corrective action is a restart of a crashed service or process. A similar mechanism is used for alerting when specific event details are sent via e-mails or SMS to responsible recipients.

For more information about Tivoli Monitoring refer to *Getting Started with Tivoli Monitoring 6.1 on Distributed Environment,* SG24-7143.

8.4.2 Mechanism of integration of Tivoli Dynamic Workload Broker with Tivoli Monitoring

In this section we explain how Tivoli Dynamic Workload Broker integrates with Tivoli Monitoring.

The integration of Tivoli Dynamic Workload Broker with Tivoli Monitoring is done by the Tivoli Monitoring Universal Agent parsing a Tivoli Dynamic Workload Broker log file. The Universal Agent periodically looks for newly appended lines in the Tivoli Dynamic Workload Broker log file. If such a line is found, it is analyzed against the definitions included in Universal Agent's metafile. If the appended line matches one of the definitions, parsed data is sent to the Tivoli Enterprise Management Server. They are then graphically presented in the Tivoli Enterprise Portal.

No situation definitions or corrective actions are provided with the default integration. However, in our scenario we demonstrate how can we create situations and set up corrective actions on them.

Figure 8-22 shows how Tivoli Dynamic Workload Broker and Tivoli Monitoring are integrated.

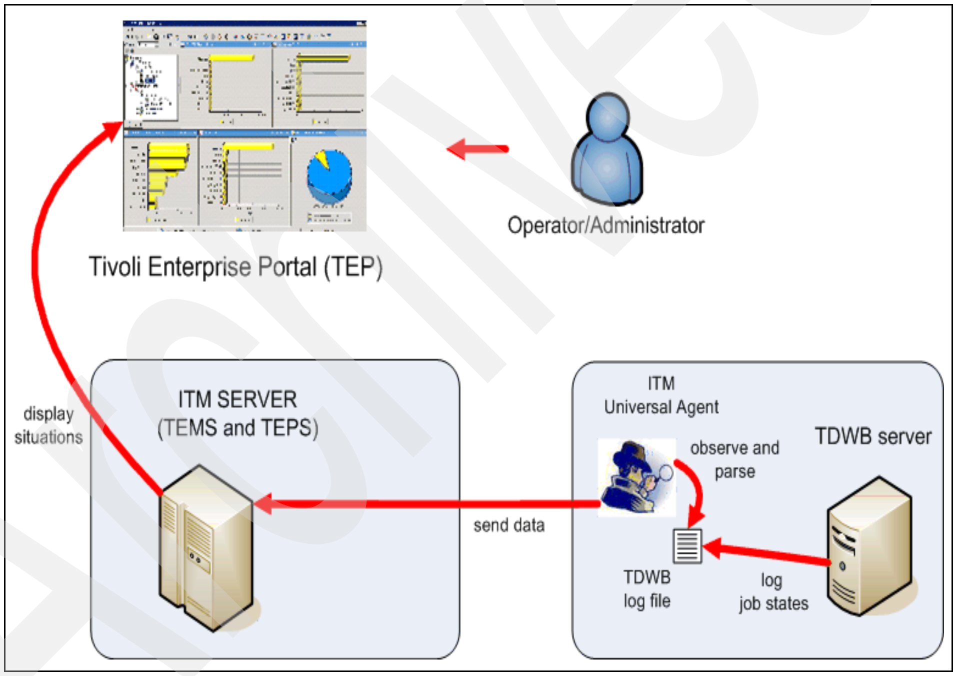


Figure 8-22 Monitoring Tivoli Dynamic Workload Broker Job states using Universal Agent

The Tivoli Dynamic Workload Broker log file contains information about states of jobs handled by the Tivoli Dynamic Workload Broker. It does not contain any information about the states of any Tivoli Dynamic Workload Broker component. The out-of-box integration does not provide any tool for monitoring of health of Tivoli Dynamic Workload Broker components.

The out-of-box integration is performed by executing the integration script. This script must be supplied by several parameters. All the necessary steps are described in 8.4.3, “Pre-integration tasks” on page 330, and 8.4.4, “Integration steps” on page 330.

8.4.3 Pre-integration tasks

In this section we describe what tasks must be done prior to performing the integration of Tivoli Dynamic Workload Broker and Tivoli Monitoring.

The Tivoli Monitoring Universal Agent must be installed on the same machine where the Tivoli Dynamic Workload Broker server resides. You must know the installation directory of the Universal Agent before you launch the integration command. If the Universal Agent is installed in its default directory, you may use the value specified in 8.4.13, “Default values and file locations” on page 391.

Choose the path where you want Tivoli Dynamic Workload Broker to create its log file. If you decide to override the default location, make sure that the directory already exists on the fleshiest. If not, create it. You do not need to set up any special access permissions for that directory because the Tivoli Dynamic Workload Broker accesses it under the account that is used for running a WebSphere Application Server. On both Windows and UNIX this account has full permissions to the fleshiest.

When you are ready to select the log file path, you must prepare the list job states that you want to write to logs and thus to be monitored by Tivoli Monitoring Universal Agent. See “List of all possible states of Tivoli Dynamic Workload Broker jobs” on page 392 for a list of possible job states.

8.4.4 Integration steps

In this section we explain how to integrate the Tivoli Dynamic Workload Broker with Tivoli Monitoring so that a monitoring of job states can be achieved.

Note: All of the scenarios included in this chapter were performed in a Windows environment. However, the steps on UNIX/Linux platforms are similar. The different default paths and commands are listed in 8.4.13, “Default values and file locations” on page 391.

The integration is performed by launching integration script tepconfig.bat. This script is located in the bin subdirectory of the Tivoli Dynamic Workload Broker installation directory.

Steps performed by the integration script

The integration script performs following tasks:

- Configures Tivoli Dynamic Workload Broker so that it logs required events to the file
- Configures the Universal Agent so that it understands the content of the Tivoli Dynamic Workload Broker log file
- Configures the views in the Tivoli Enterprise Portal so that it is able to display situations that originate from the Tivoli Dynamic Workload Broker environment

Be sure that you have prepared all the necessary inputs, as stated in 8.4.3, "Pre-integration tasks" on page 330. If you have all of the data ready, you can start the integration.

Complete list of steps

In this section we provide the complete list of steps that must be done either on the Tivoli Dynamic Workload Broker side or on the Tivoli Enterprise Portal side to complete the integration.

The complete list of steps is included below:

1. Prepare all the necessary data, as described in "Pre-integration tasks" on page 330.
2. Source the Tivoli Dynamic Workload Broker command-line environment.
3. Issue the integration script.
4. Recycle the Tivoli Dynamic Workload Broker's WebSphere Application server.
5. Check the log file presence.
6. Add additional settings on the Tivoli Monitoring side, as described in "Configuring Universal Agent to accept a FILE data provider" on page 337:
   - Configuring Tivoli Monitoring Universal Agent, so it can analyze text files
   - Creating situations
   - Setting thresholds
   - Creating corrective actions

Sourcing the Tivoli Dynamic Workload Broker command-line environment

Open a command window and go to the directory C:\Program Files\IBM\ITDWB\server\bin and issue **`tdwb_env`**.

Now you have set up all of the necessary environmental variables and you can launch the integration script itself. Before launching the integration script, read this chapter to its end. There is a difference between results that can you get when you run the script with the default parameters, and when you specify additional parameters. Essentially, the default integration sets only some of possible job states to monitoring.

Launching the integration script

Remain in the directory C:\Program Files\IBM\ITDWB\server\bin and issue **`tepconfig.bat -UAInstDir`** in the TM_installation_directory.

This command performs the integration with the default values. See 8.4.13, “Default values and file locations” on page 391 to find out where the Tivoli Dynamic Workload Broker log file will be created and what events will be logged into it.

> **Important:** On Windows systems you can receive an information such as `C:\Program' is not recognized as an internal or external command, operable program or batch file.` In Tivoli Dynamic Workload Broker 1.1, this message is caused by an internal defect in the tepconfig.bat file. An internal defect 30542 has been opened and this problem will be fixed in future releases. However, if you only have Tivoli Dynamic Workload Broker V1.1, you may fix this defect by yourself. For more information about this issue see 10.3.2, “Problems with running the integration script on Windows” on page 505.

For additional troubleshooting issues related to integration of Tivoli Dynamic Workload Broker with Tivoli Monitoring, refer to 10.3, “Troubleshooting the integration with IBM Tivoli Monitoring” on page 504.

In most cases you would like to change the directory where the log file will be created and you would also like to explicitly specify what job states should be written to the log file. In these cases you will have to specify additional parameters. The names of parameters are case sensitive.

- -eventFilePathName

  Provide the full directory path for where you want the Tivoli Dynamic Workload Broker to create the log file. Do not include the file name in the path. The structure of the file name is fixed and cannot be overridden.

Important: The value of “eventFilePathName” can be upredictably parsed on a Windows platform in Tivoli Dynamic Workload Broker v.1.11.1. Some characters preceded by a backslash (\) are wrongly interpreted as special characters. Typical example is \t, which is treated as TAB-sign. Because of this behavior, it is better to use double backslashes (\\) while specifying the path. An internal defect 30643 has been opened and this problem will be fixed in future releases. See 10.3.3, “Wrongly interpreted characters in log file path on Windows” on page 506, for more details about this issue.

- -events

  The value for this parameter is a list of job states that should be logged into the log file. Values must be separated by spaces.

Important: When launching the integration script and specifying multiple event types (using -events argument) together with additional parameters, such as -metafileName or -UAApname, the integration run time is not able to parse more than one event type. The integration script abends with ean rror message about improperly supplied parameters. An internal defect 31336 has been opened for this problem. This error affects only Tivoli Dynamic Workload Broker v.1.1 and will be fixed in future releases. We include a workaround for this issue in 10.3.4, “Cannot specify multiple event types together with parameters” on page 506.

See the following examples of launching the integration script under various circumstances. Do not forget to restart WebSphere after any change you perform for the integration.

Example 8-2 shows how to launch an integration script with default values. Note that the integration has two phases:

1. Tivoli Dynamic Workload Broker configures its *TEPListener* component. This is a server component that observes the states of jobs. Based on the TEPListener configuration, defined job states are written to the log file.
2. The Tivoli Monitoring Universal Agent is instructed to load the metafile with monitoring definitions (messages beginning with “KUMP”).

Example 8-2 Issuing an integration script on Windows with default values

```
C:\Program Files\IBM\ITDWB\Server\bin>tepconfig.bat -UAInstDir
C:\IBM\ITM
Mar 3, 2007 1:24:00 AM
com.ibm.scheduling.jobdispatcher.jobstatuslistener.TEPJobStatusChangeLi
stenerProperties setEventFileName
```

```
INFO: AWKTEP013I TEP listener has been configured to write events to
C:\Program Files\IBM\ITDWB\Server\logs
KUMPS001I Console input accepted.
KUMPS020I Import successfully completed for
C:\Program Files\IBM\ITDWB\Server\TEP\TDWB11Meta.mdl
```

Example 8-3 shows how the default log file path can be overridden. We also used a custom name for the Universal Agent definitions metafile.

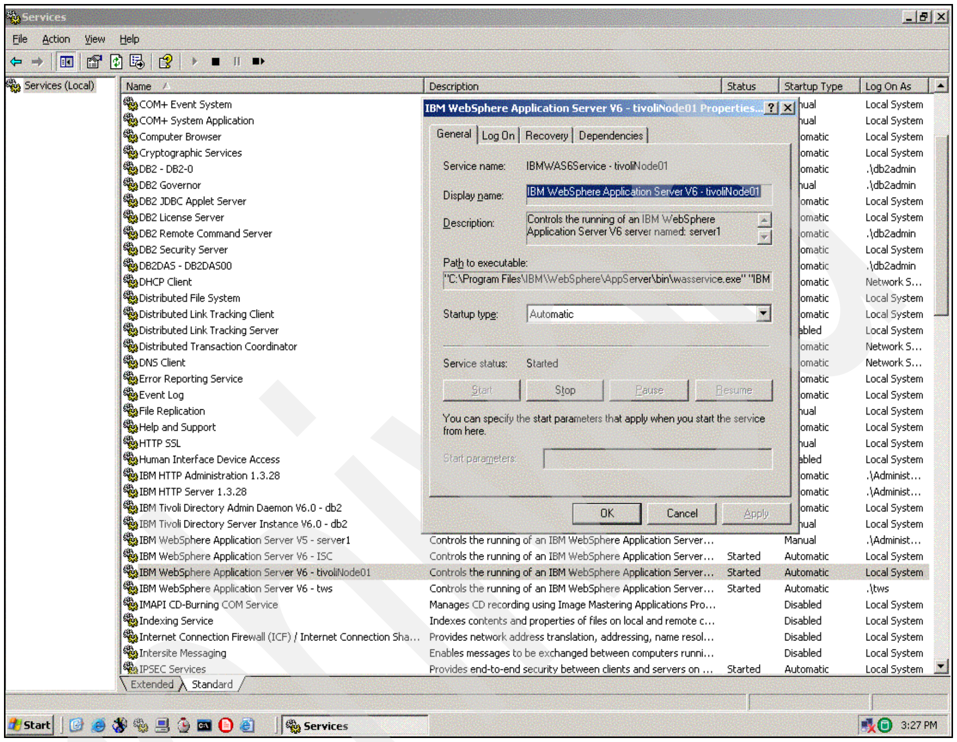
Example 8-3 Issuing an integration script on Windows with non-default values

```
C:\Program Files\IBM\ITDWB\Server\bin>tepconfig.bat -UAInstDir
C:\IBM\ITM -eventFilePathName C:\IBM\ITM\logs -metafileName
c:\IBM\itm\logs\meta.mdl

Mar 8, 2007 5:50:00 PM
com.ibm.scheduling.jobdispatcher.jobstatuslistener.TEPJobStatusChangeLi
stenerProperties setEventFileName
INFO: AWKTEP013I TEP listener has been configured to write events to
C:\IBM\logs
KUMPS001I Console input accepted.
KUMPS020I Import successfully completed for c:\IBM\itm\logs\meta.mdl
```

Recycling Tivoli Dynamic Workload Broker's WebSphere Application server

You must restart the WebSphere server for the changes to take effect. On Windows, go to **Start** → **Control Panel** → **Administrative Tools** → **Services**, find the appropriate IBM WebSphere Application server, and click **Restart**.

The name of the WebSphere where Tivoli Dynamic Workload Broker is installed by default contains the value of nodename, which corresponds to the value of the node parameter that was specified during the installation. Figure 8-23 shows how we can determine the correct service to restart.



IBM WebSphere Application Server V6 - tivoliNode01 Properties...

General | Log On | Recovery | Dependencies

Service name: IBMWAS6Service - tivoliNode01

Display name: IBM WebSphere Application Server V6 - tivoliNode01

Description: Controls the running of an IBM WebSphere Application Server V6 server named: server1

Path to executable:
"C:\Program Files\IBM\WebSphere\AppServer\bin\wasservice.exe" "IBM

Startup type: Automatic

Service status: Started

Start | Stop | Pause | Resume

You can specify the start parameters that apply when you start the service from here.

Start parameters:

OK | Cancel | Apply

Name	Description	Status	Startup Type	Log On As
COM+ Event System			nual	Local System
COM+ System Application			nual	Local System
Computer Browser			omatic	Local System
Cryptographic Services			omatic	Local System
DB2 - DB2-0			omatic	.\db2admin
DB2 Governor			nual	.\db2admin
DB2 JDBC Applet Server			omatic	Local System
DB2 License Server			omatic	Local System
DB2 Remote Command Server			omatic	.\db2admin
DB2 Security Server			omatic	Local System
DB2DAS - DB2DAS00			omatic	.\db2admin
DHCP Client			omatic	Network S...
Distributed File System			omatic	Local System
Distributed Link Tracking Client			omatic	Local System
Distributed Link Tracking Server			abled	Local System
Distributed Transaction Coordinator			omatic	Network S...
DNS Client			omatic	Network S...
Error Reporting Service			omatic	Local System
Event Log			omatic	Local System
File Replication			nual	Local System
Help and Support			omatic	Local System
HTTP SSL			nual	Local System
Human Interface Device Access			abled	Local System
IBM HTTP Administration 1.3.28			omatic	.\Administ...
IBM HTTP Server 1.3.28			omatic	.\Administ...
IBM Tivoli Directory Admin Daemon V6.0 - db2			omatic	Local System
IBM Tivoli Directory Server Instance V6.0 - db2			nual	Local System
IBM WebSphere Application Server V5 - server1	Controls the running of an IBM WebSphere Application Server...		Manual	.\Administ...
IBM WebSphere Application Server V6 - ISC	Controls the running of an IBM WebSphere Application Server...	Started	Automatic	Local System
IBM WebSphere Application Server V6 - tivoliNode01	Controls the running of an IBM WebSphere Application Server...	Started	Automatic	Local System
IBM WebSphere Application Server V6 - tws	Controls the running of an IBM WebSphere Application Server...	Started	Automatic	.\tws
IMAPI CD-Burning COM Service	Manages CD recording using Image Mastering Applications Pro...		Disabled	Local System
Indexing Service	Indexes contents and properties of files on local and remote c...		Disabled	Local System
Internet Connection Firewall (ICF) / Internet Connection Sha...	Provides network address translation, addressing, name resol...		Disabled	Local System
Intersite Messaging	Enables messages to be exchanged between computers runni...		Disabled	Local System
IPSEC Services	Provides end-to-end security between clients and servers on ...	Started	Automatic	Local System

Figure 8-23   Selecting the correct WebSphere Application Server to restart

Checking the Tivoli Dynamic Workload Broker log file existence

After you have restarted the WebSphere Application Server, the TEPListener component is activated inside the Tivoli Dynamic Workload Broker server. TEPListener then waits for occurrences of defined events. If any defined activities occur when processing Tivoli Dynamic Workload Broker jobs, they are logged into the file.

The log file can be found either in the default location or in the location that you have specified with the -eventFilePathName parameter. The name of the file is as follows:

```
TEPEVENTyyyymmddHHMM.log
```

For example, a log file that has been created on the 8th of March 2007 at 6:05 p.m. is named:

```
TEPEVENT200703081805.log
```

The Tivoli Dynamic Workload Broker events are logged to this file until it reaches its defined maximum size. After that a new log file is created with an appropriate file name (that includes a new time stamp). The Tivoli Dynamic Workload Broker then continues to log to the new file.

Note: The log file does not get created until an defined event inside the Tivoli Dynamic Workload Broker occurs. For instance, if you launched the integration script with the -event CANCEL parameter, only cancelled job instances are reported to the log file. Until a job cancellation occurs, a log file does not get created.

However, if a log file does not appear to be in the correct directory, even if a configured event has already occurred within the Tivoli Dynamic Workload Broker, something is probably wrong. See 10.3.6, “Tivoli Dynamic Workload Broker log file not created” on page 508 for more troubleshooting techniques.

Next steps

Now you have finished the out-of-box integration. However, no views, situations, or corrective actions are set up on the Tivoli Monitoring side. See 8.4.6, “Configuring Universal Agent to accept a FILE data provider” on page 337 to learn how to take additional steps that must be performed on the Tivoli Monitoring side. Then follow the instructions described in 8.4.8, “Creating a view on monitored data” on page 346, 8.4.9, “Setting up thresholds” on page 349, 8.4.10, “Creating situations” on page 355, and 8.4.11, “Setting up automatic corrective action” on page 357.

8.4.5 Changing the integration criteria at a later time

It is possible to change the integration criteria after the integration has been performed and Universal Agent has already started to observe the Tivoli Dynamic Workload Broker log file.

The reason for a change can be one of the following:

- The log file path needs to be changed (this is usually not necessary, but sometimes there is a requirement to place all of the log files into one specified directory tree).
- The Tivoli Monitoring Universal Agent was reinstalled.
- The scope of monitored job states needs to be adjusted.

The integration script performs the necessary steps for you, including the refresh of those settings that are already registered within the Universal Agent.

Important: When issuing the integration script for a second (or any further) time and redefining the list of event types, only new required event types are added, but the old unwanted event types are *not removed*. This is an internal error. An internal defect 31351 was opened for this problem. This error affects only Tivoli Dynamic Workload Broker v.1.1 and will be fixed in future releases. We include a workaround for this issue in 10.3.5, “Cannot remove unwanted event types” on page 507.

If you have run the integration script again and have adjusted the integration settings you must restart the WebSphere Application Server under which Tivoli Dynamic Workload Broker is installed. The changes take effect only after the restart of the WebSphere Application Server.

8.4.6 Configuring Universal Agent to accept a FILE data provider

In this section we describe the necessary steps that must be taken to configure Universal Agent to analyze text log files.

Even if the Tivoli Monitoring Universal Agent should by default accept text log files as its data provider, we have experienced different behavior in our scenarios. Unless we explicitly specified a FILE data provider in the configuration of Universal Agent, we did not see any input from the Tivoli Dynamic Workload Broker server.

If you are able to see the TDWB application in Tivoli Enterprise Portal you may skip this section. However, you can read here how to adjust the frequency of how often the Tivoli Dynamic Workload Broker log file should be checked by the Universal Agent.

In the opposite case (you do not see the TDWB application within the Universal Agent branch) you must do all of the steps described in this section.

Important: The steps listed in this section must be performed on the system where the Tivoli Dynamic Workload Broker is installed. We must configure the Universal Agent, which is running on the same machine where the Tivoli Dynamic Workload Broker log file was created.

1. Launch the Manage Tivoli Monitoring Services window by clicking **Start** → **Programs** → **Tivoli Monitoring** → **Manage Tivoli Monitoring Services** (Figure 8-24).

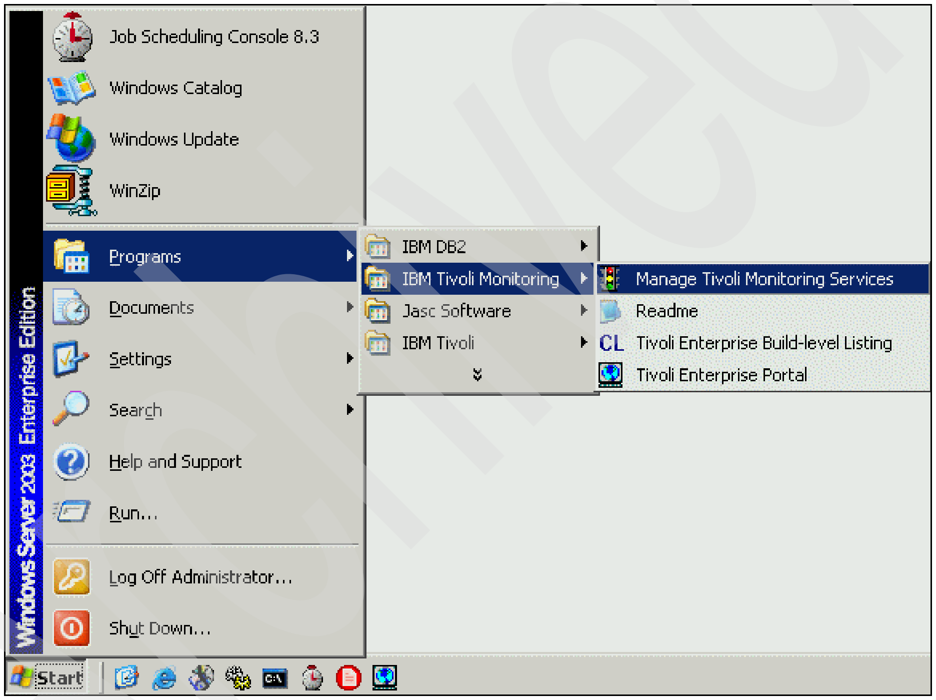
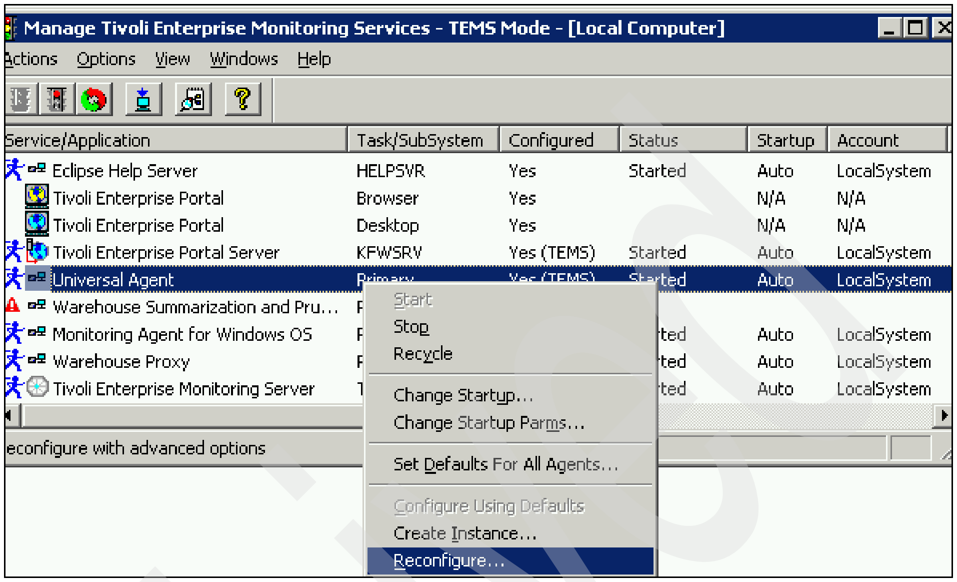


Figure 8-24 Launching Manage Tivoli Monitoring Services

2. In Manage Tivoli Monitoring Services right-click the agent and select **Reconfigure** (Figure 8-25).

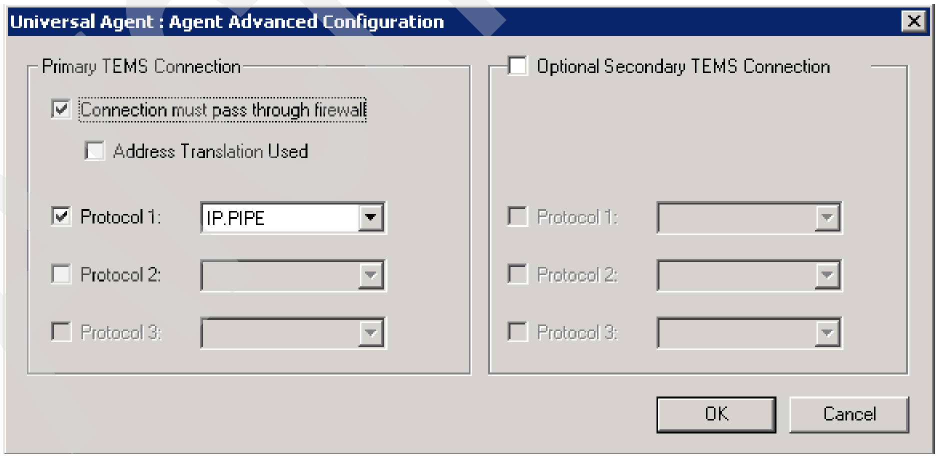

Manage Tivoli Enterprise Monitoring Services - TEMS Mode - [Local Computer]

Service/Application	Task/SubSystem	Configured	Status	Startup	Account
Eclipse Help Server	HELPSVR	Yes	Started	Auto	LocalSystem
Tivoli Enterprise Portal	Browser	Yes		N/A	N/A
Tivoli Enterprise Portal	Desktop	Yes		N/A	N/A
Tivoli Enterprise Portal Server	KFWSRV	Yes (TEMS)	Started	Auto	LocalSystem
Universal Agent	Primary	Yes (TEMS)	Started	Auto	LocalSystem
Warehouse Summarization and Pru...	P				
Monitoring Agent for Windows OS	P		ted	Auto	LocalSystem
Warehouse Proxy	P		ted	Auto	LocalSystem
Tivoli Enterprise Monitoring Server	T		ted	Auto	LocalSystem

econfigure with advanced options

Figure 8-25 Reconfiguring Universal Agent - step 1

3. Click **OK** in the following two windows (Figure 8-26 and Figure 8-27 on page 340).

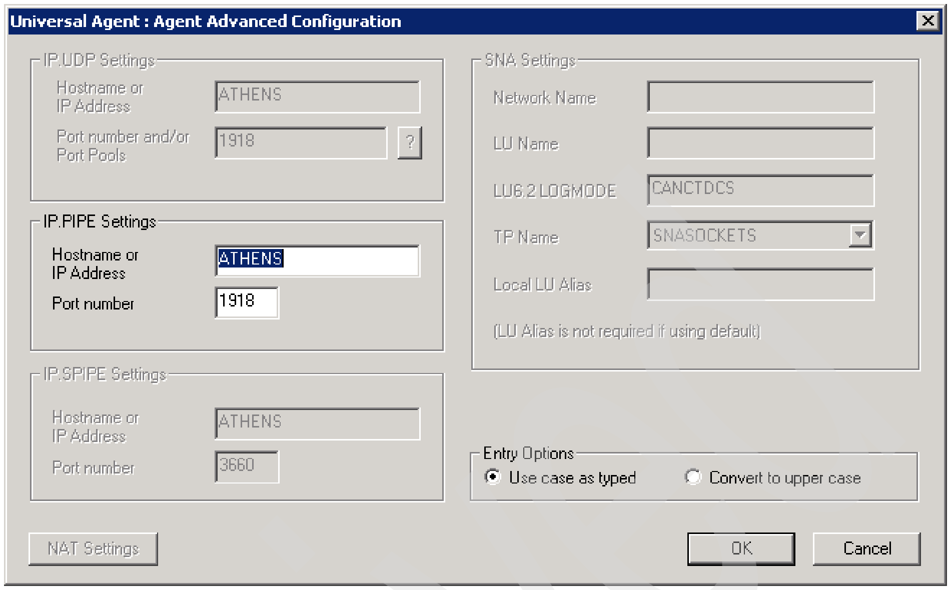

Universal Agent : Agent Advanced Configuration

Primary TEMS Connection

- [x] Connection must pass through firewall
- [ ] Address Translation Used
- [x] Protocol 1: IP.PIPE
- [ ] Protocol 2:
- [ ] Protocol 3:

- [ ] Optional Secondary TEMS Connection
- [ ] Protocol 1:
- [ ] Protocol 2:
- [ ] Protocol 3:

Figure 8-26 Reconfiguring Universal Agent - step 2


Universal Agent : Agent Advanced Configuration

IP.UDP Settings
Hostname or IP Address: ATHENS
Port number and/or Port Pools: 1918

IP.PIPE Settings
Hostname or IP Address: ATHENS
Port number: 1918

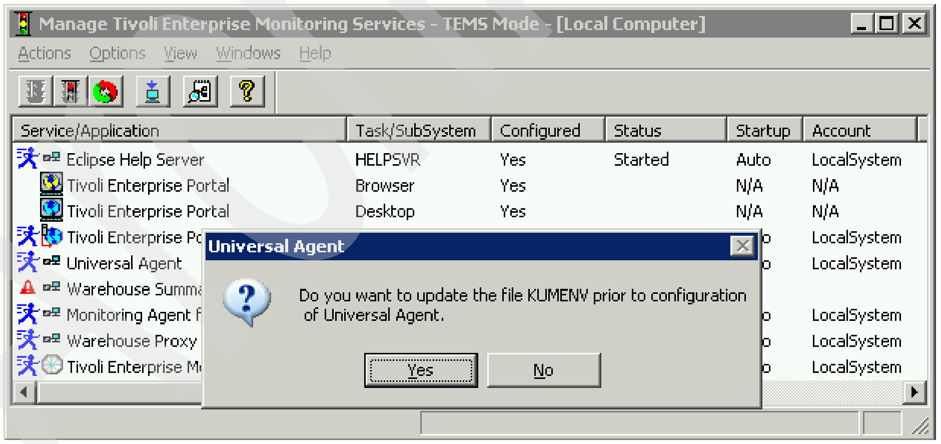
IP.SPIPE Settings
Hostname or IP Address: ATHENS
Port number: 3660

SNA Settings
Network Name:
LU Name:
LU6.2 LOGMODE: CANCTDCS
TP Name: SNASOCKETS
Local LU Alias:
(LU Alias is not required if using default)

Entry Options
[✓ Use case as typed] Convert to upper case

Figure 8-27 Reconfiguring Universal Agent - step 3

4. Click **Yes** in the following window, because we want to change Universal Agent's startup environment (Figure 8-28).


Manage Tivoli Enterprise Monitoring Services - TEMS Mode - [Local Computer]

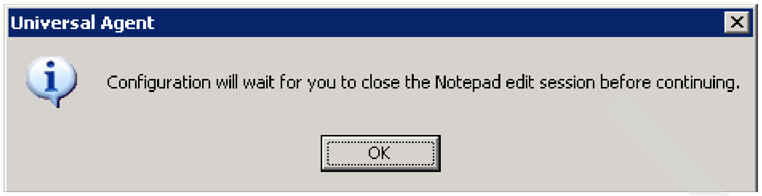
Service/Application	Task/SubSystem	Configured	Status	Startup	Account
Eclipse Help Server	HELPSVR	Yes	Started	Auto	LocalSystem
Tivoli Enterprise Portal	Browser	Yes		N/A	N/A
Tivoli Enterprise Portal	Desktop	Yes		N/A	N/A
Tivoli Enterprise Po				o	LocalSystem
Universal Agent				o	LocalSystem
Warehouse Summa					
Monitoring Agent f				o	LocalSystem
Warehouse Proxy				o	LocalSystem
Tivoli Enterprise M				o	LocalSystem

Universal Agent

Do you want to update the file KUMENV prior to configuration of Universal Agent.

Figure 8-28 Reconfiguring Universal Agent - step 4

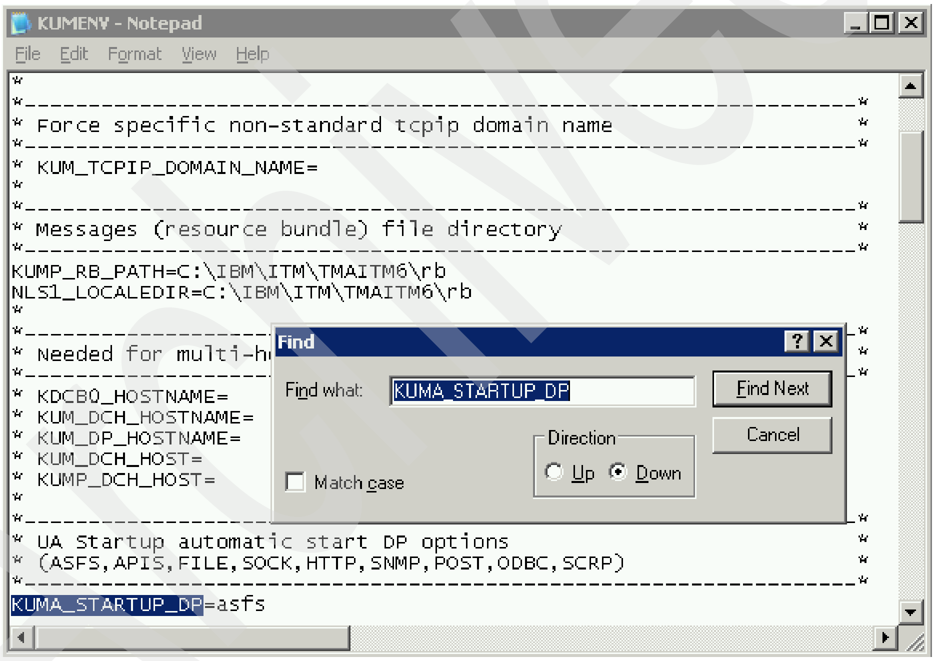
5. Click **OK** in the following window. After that the notepad window will pop-up (Figure 8-29).



Configuration will wait for you to close the Notepad edit session before continuing.

Figure 8-29 Reconfiguring Universal Agent - step 5

6. In the notepad window, search for text KUMA_STARTUP_DP (Figure 8-30). This line tells the Universal Agent which data providers he uses when collecting data from monitored resources.



```
*
*-------------------------------------------------------------------*
* Force specific non-standard tcpip domain name                     *
*-------------------------------------------------------------------*
* KUM_TCPIP_DOMAIN_NAME=
*
*-------------------------------------------------------------------*
* Messages (resource bundle) file directory                         *
*-------------------------------------------------------------------*
KUMP_RB_PATH=C:\IBM\ITM\TMAITM6\rb
NLS1_LOCALEDIR=C:\IBM\ITM\TMAITM6\rb
*
*-------------------------------------------------------------------*
* Needed for multi-h                                                *
*-------------------------------------------------------------------*
* KDCB0_HOSTNAME=
* KUM_DCH_HOSTNAME=
* KUM_DP_HOSTNAME=
* KUM_DCH_HOST=
* KUMP_DCH_HOST=
*
*-------------------------------------------------------------------*
* UA Startup automatic start DP options                             *
* (ASFS,APIS,FILE,SOCK,HTTP,SNMP,POST,ODBC,SCRP)                     *
*-------------------------------------------------------------------*
KUMA_STARTUP_DP=asfs
```

Find what: KUMA_STARTUP_DP
Direction: Up [✓ Down]

Figure 8-30 Reconfiguring Universal Agent - step 6

By default, it contains the value ASFS, which also includes the FILE data provider. However, in our scenario we had to explicitly specify the FILE data provider. To do this, separate the last value of KUMA_STARTUP_DP with a comma (,) and add FILE at the end of the line.

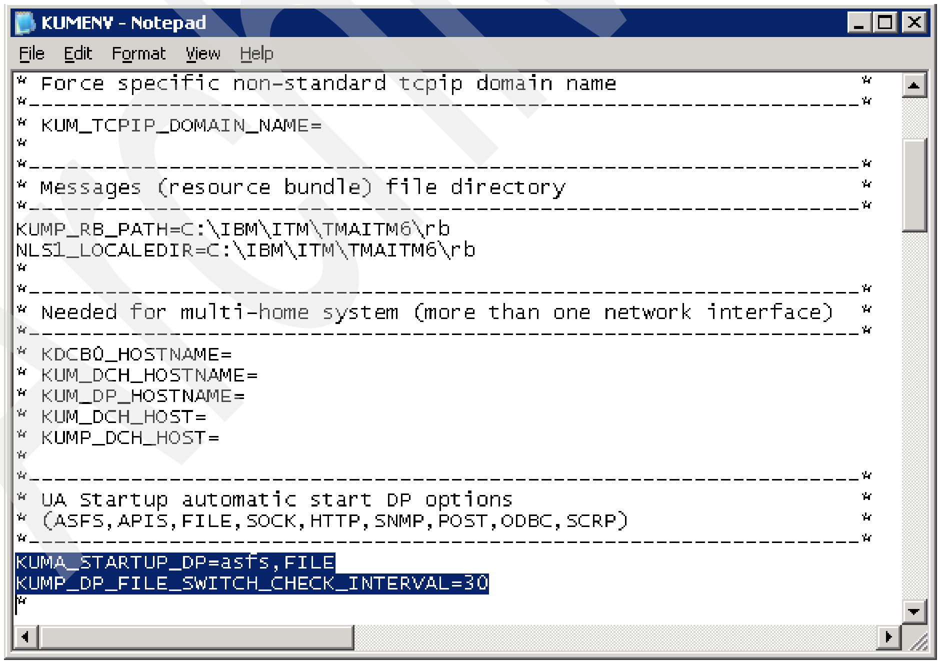
Search for the line containing KUMP_DP_FILE_SWITCH_CHECK_INTERVAL. If this line is included in the config file, adjust it to the value you consider reasonable. The units for this value are in *seconds*. If this variable is not included in the configuration file, Universal Agent uses the default value, which is 10 minutes. This means that Universal Agent checks for newly appended lines into log files every 10 minutes. For the purposes of job management this interval can be too long, because usually it is good to know about failed Tivoli Dynamic Workload Broker jobs as soon as possible.

Tip: We recommend that the value for the variable KUMP_DP_FILE_SWITCH_CHECK_INTERVAL is 30 seconds.

7. At this point we should naje two essential changes to the Universal Agent environment:
   - Add the file data provider.
   - Modify the file checking interval. Added (modified) lines should look like this:

     ```
     KUMA_STARTUP_DP=asfs,FILE
     KUMP_DP_FILE_SWITCH_CHECK_INTERVAL=30
     ```

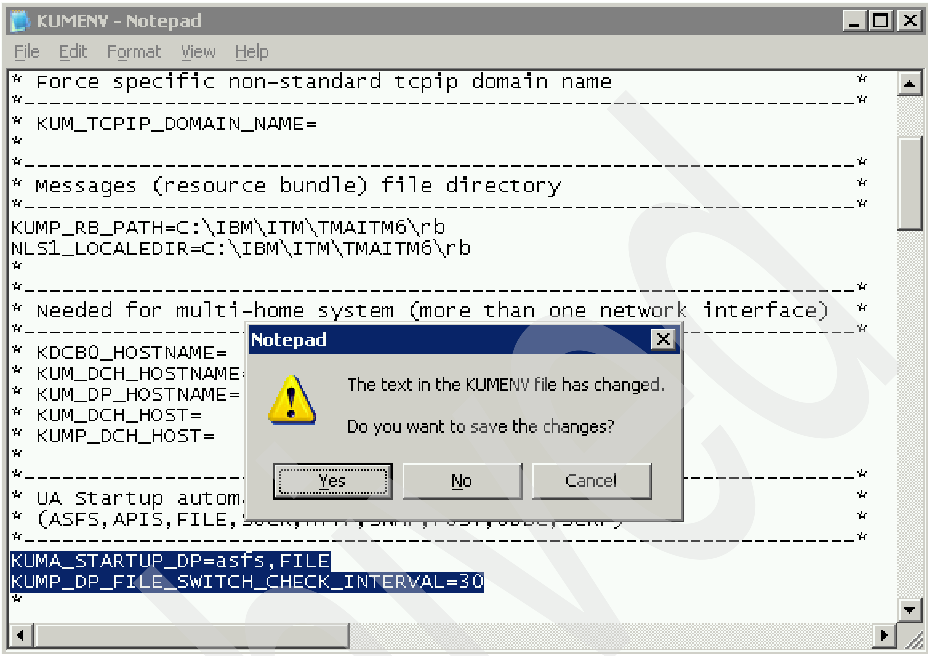
   See Figure 8-31.



```
* Force specific non-standard tcpip domain name                    *
*------------------------------------------------------------------*
* KUM_TCPIP_DOMAIN_NAME=
*
*------------------------------------------------------------------*
* Messages (resource bundle) file directory                        *
*------------------------------------------------------------------*
KUMP_RB_PATH=C:\IBM\ITM\TMAITM6\rb
NLS1_LOCALEDIR=C:\IBM\ITM\TMAITM6\rb
*
*------------------------------------------------------------------*
* Needed for multi-home system (more than one network interface)   *
*------------------------------------------------------------------*
* KDCB0_HOSTNAME=
* KUM_DCH_HOSTNAME=
* KUM_DP_HOSTNAME=
* KUM_DCH_HOST=
* KUMP_DCH_HOST=
*
*------------------------------------------------------------------*
* UA Startup automatic start DP options                            *
* (ASFS,APIS,FILE,SOCK,HTTP,SNMP,POST,ODBC,SCRP)                   *
*------------------------------------------------------------------*
KUMA_STARTUP_DP=asfs,FILE
KUMP_DP_FILE_SWITCH_CHECK_INTERVAL=30
*
```

Figure 8-31 Reconfiguring Universal Agent - step 7

8. Close the Notepad window and save the changes made to the configuration file (Figure 8-32).



```
* Force specific non-standard tcpip domain name                              *
*----------------------------------------------------------------------------*
* KUM_TCPIP_DOMAIN_NAME=
*
*----------------------------------------------------------------------------*
* Messages (resource bundle) file directory                                  *
*----------------------------------------------------------------------------*
KUMP_RB_PATH=C:\IBM\ITM\TMAITM6\rb
NLS1_LOCALEDIR=C:\IBM\ITM\TMAITM6\rb
*
*----------------------------------------------------------------------------*
* Needed for multi-home system (more than one network interface)             *
*----------------------------------------------------------------------------*
* KDCB0_HOSTNAME=
* KUM_DCH_HOSTNAME=
* KUM_DP_HOSTNAME=
* KUM_DCH_HOST=
* KUMP_DCH_HOST=
*
*----------------------------------------------------------------------------*
* UA Startup autom
* (ASFS,APIS,FILE,
*----------------------------------------------------------------------------*
KUMA_STARTUP_DP=asfs,FILE
KUMP_DP_FILE_SWITCH_CHECK_INTERVAL=30
*
```

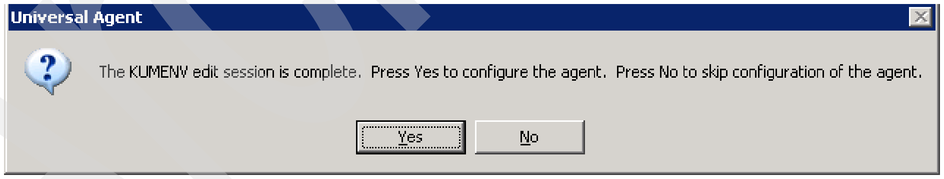
Notepad

The text in the KUMENV file has changed.

Do you want to save the changes?

Figure 8-32 Reconfiguring Universal Agent - step 8

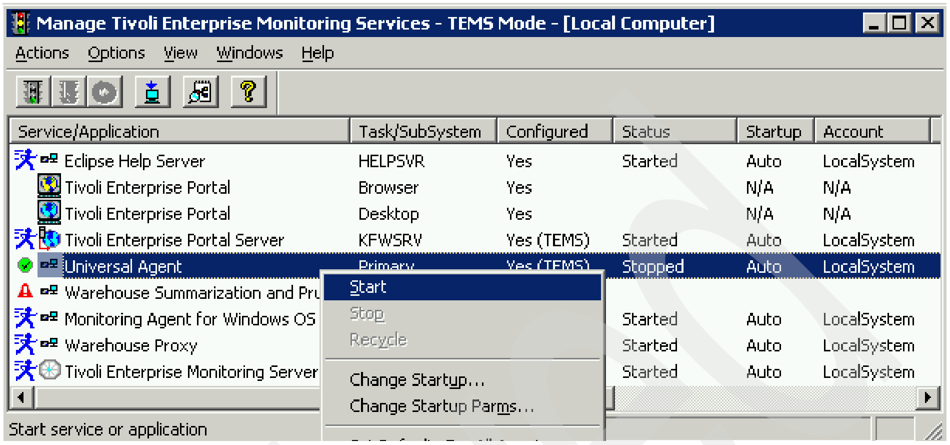
9. Click **Yes** in the following window. This will bring you back to Manage Tivoli Monitoring Services window (Figure 8-33).



The KUMENV edit session is complete. Press Yes to configure the agent. Press No to skip configuration of the agent.

Figure 8-33 Reconfiguring Universal Agent - step 9

10. Start the Universal Agent that was stopped prior to us starting to edit its configuration (Figure 8-34).


Manage Tivoli Enterprise Monitoring Services - TEMS Mode - [Local Computer]
Actions
Options
View
Windows
Help
Start
Stop
Recycle
Change Startup...
Change Startup Parms...
Start service or application

Service/Application	Task/SubSystem	Configured	Status	Startup	Account
Eclipse Help Server	HELPSVR	Yes	Started	Auto	LocalSystem
Tivoli Enterprise Portal	Browser	Yes		N/A	N/A
Tivoli Enterprise Portal	Desktop	Yes		N/A	N/A
Tivoli Enterprise Portal Server	KFWSRV	Yes (TEMS)	Started	Auto	LocalSystem
Universal Agent	Primary	Yes (TEMS)	Stopped	Auto	LocalSystem
Warehouse Summarization and Pru					
Monitoring Agent for Windows OS			Started	Auto	LocalSystem
Warehouse Proxy			Started	Auto	LocalSystem
Tivoli Enterprise Monitoring Server			Started	Auto	LocalSystem

Figure 8-34 Reconfiguring Universal Agent - step 10

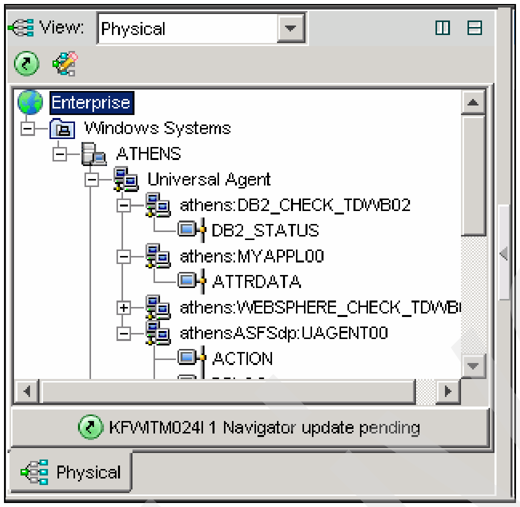
You have activated the FILE data provider for Universal Agent.

8.4.7 Viewing the application in the Tivoli Enterprise Portal

In previous sections we took the following actions:

- Launched an integration script, which performed these tasks:
  - Configured the TEPListener component of the ITDWB enterprise application
  - Imported the definition metafile into Universal Agent
- Restarted WebSphere Application Server where the Tivoli Dynamic Workload Broker is installed and thus activated the TEPListener component
- Configured Universal Agent to be able to use the FILE data provider

The result of these actions should be visible in the Tivoli Enterprise Portal. If you performed these actions while the Tivoli Enterprise Portal was running, you can see a message showing that the Navigator pane needs to be updated (Figure 8-35).



KFWITM024I 1 Navigator update pending

Figure 8-35 Navigator pane - refresh pending

After you have refreshed the Navigator pane, you can see that a new application has been added. The name of the new application should consist of the of the Tivoli Dynamic Workload Broker server host name and the application name specified in the Universal Agent’s metafile. The two-digit suffix displays the application version and modification number (initial value is 00) (Figure 8-36).

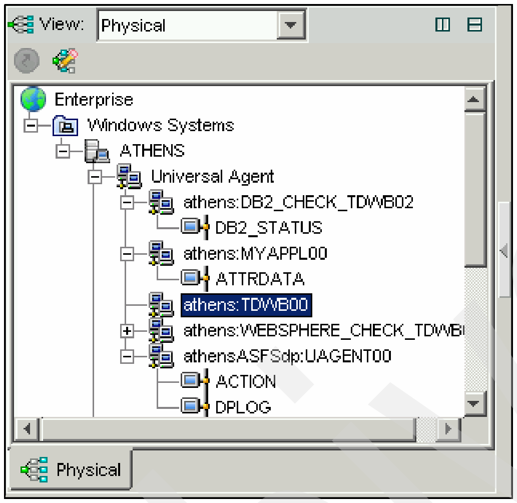


Figure 8-36   Navigator pane - refreshed, new application added

8.4.8 Creating a view on monitored data

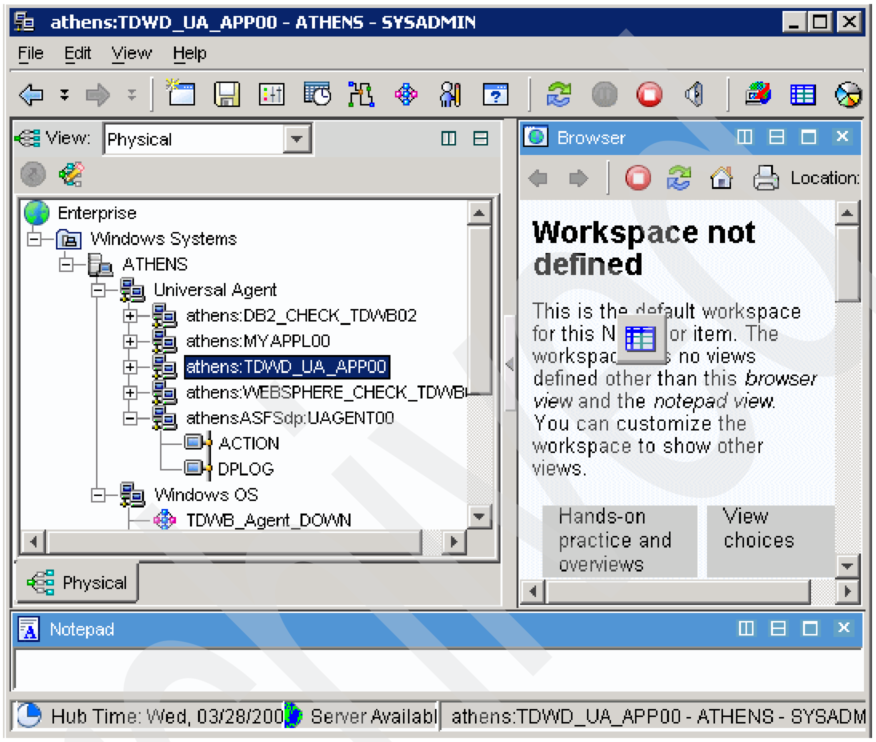
In this section we describe how to create a view that displays the monitored data.

A *view* displays the monitored data in the way we choose. It may be a table, one of many types of graphs, or whatever Tivoli Enterprise Portal offers.

For our purposes a *table view* is sufficient. Defining that view requires a few steps:

1. In the Navigator pane click our application.

2. Click the table icon in the top of the Tivoli Enterprise Portal desktop. Your cursor changes and looks like the icon you just clicked. Move the cursor over the undefined workspace and click. Answer **Yes** if a question window “Assign the query now?” appears (Figure 8-37).



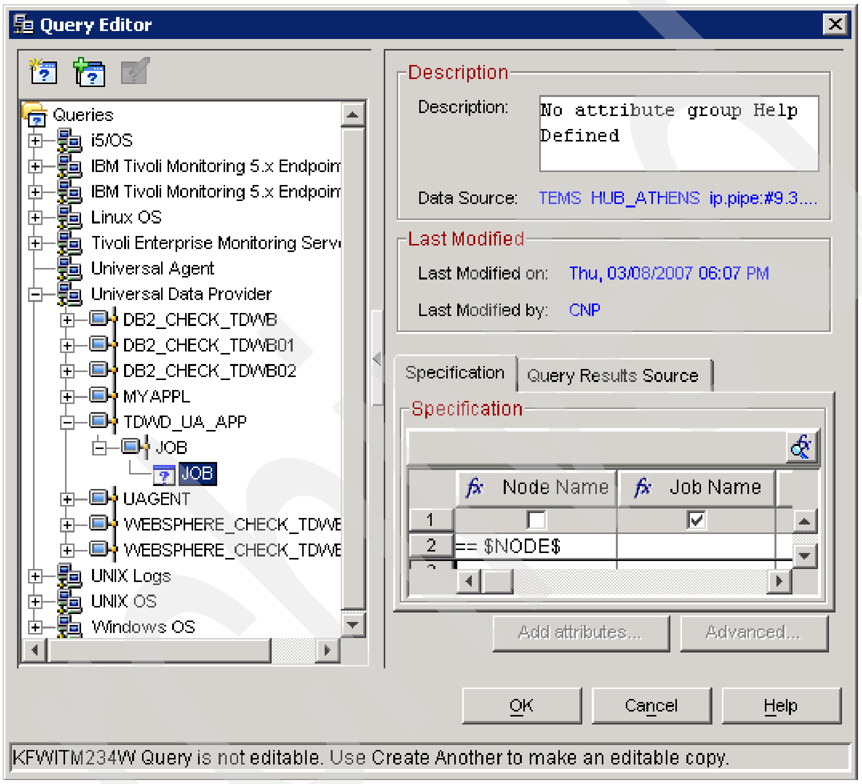
Workspace not defined

This is the default workspace for this N... or item. The workspac... s no views defined other than this *browser view* and the *notepad view*. You can customize the workspace to show other views.

Figure 8-37 Creating a view - step 1

3. In the following window click **Click here to assign the query**.

4. The Query Editor window appears. In the left pane expand the **Universal Data Provider** branch. After that expand our application branch. Then expand the attribute group branch and navigate to the deepest level of this branch. The right pane shows all of the attributes defined within attribute group of our application. Select or deselect the attributes so that you customize what attributes you want to have your view (Figure 8-38).



Description: No attribute group Help Defined

Data Source: TEMS HUB_ATHENS ip.pipe:#9.3....

Last Modified on: Thu, 03/08/2007 06:07 PM

Last Modified by: CNP

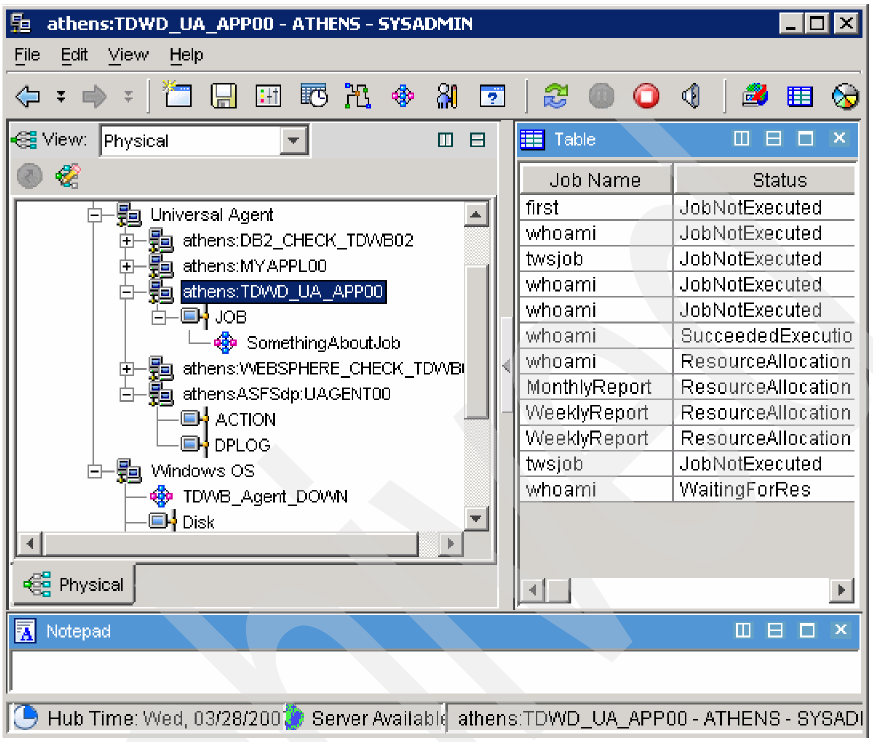
	Node Name	Job Name
1	☐	☑
2	== $NODE$	

KFWITM234W Query is not editable. Use Create Another to make an editable copy.

Figure 8-38 Creating a view - step 2

5. Click **OK** in the Query Editor window. Also confirm the following window.

You have created a view displaying the monitored data. Figure 8-39 shows the defined view.



- Universal Agent
  - athens:DB2_CHECK_TDWB02
  - athens:MYAPPL00
  - athens:TDWD_UA_APP00
    - JOB
      - SomethingAboutJob
  - athens:WEBSPHERE_CHECK_TDWB
  - athensASFSdp:UAGENT00
    - ACTION
    - DPLOG
- Windows OS
  - TDWB_Agent_DOWN
  - Disk

Job Name	Status
first	JobNotExecuted
whoami	JobNotExecuted
twsjob	JobNotExecuted
whoami	JobNotExecuted
whoami	JobNotExecuted
whoami	SucceededExecutio
whoami	ResourceAllocation
MonthlyReport	ResourceAllocation
WeeklyReport	ResourceAllocation
WeeklyReport	ResourceAllocation
twsjob	JobNotExecuted
whoami	WaitingForRes

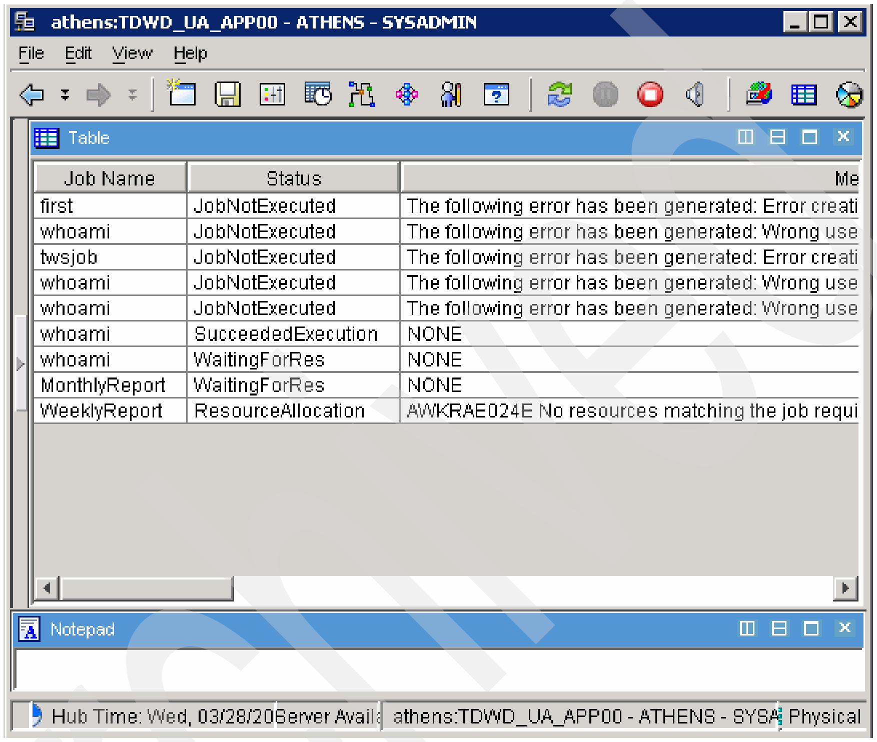
Hub Time: Wed, 03/28/200 Server Availabl athens:TDWD_UA_APP00 - ATHENS - SYSADI

Figure 8-39 Tivoli Enterprise Portal - Newly created view

8.4.9 Setting up thresholds

In this section we describe how to set up thresholds distinguishing displayed data by severity.

In a newly created view you can see the monitored data, but they are not distinguished by severity. Tivoli Monitoring server does not know yet which data carry information about something harmless, and which data report serious errors. Figure 8-40 shows the example of the newly created view that has no thresholds set up.



Job Name	Status	Me
first	JobNotExecuted	The following error has been generated: Error creati
whoami	JobNotExecuted	The following error has been generated: Wrong use
twsjob	JobNotExecuted	The following error has been generated: Error creati
whoami	JobNotExecuted	The following error has been generated: Wrong use
whoami	JobNotExecuted	The following error has been generated: Wrong use
whoami	SucceededExecution	NONE
whoami	WaitingForRes	NONE
MonthlyReport	WaitingForRes	NONE
WeeklyReport	ResourceAllocation	AWKRAE024E No resources matching the job requi

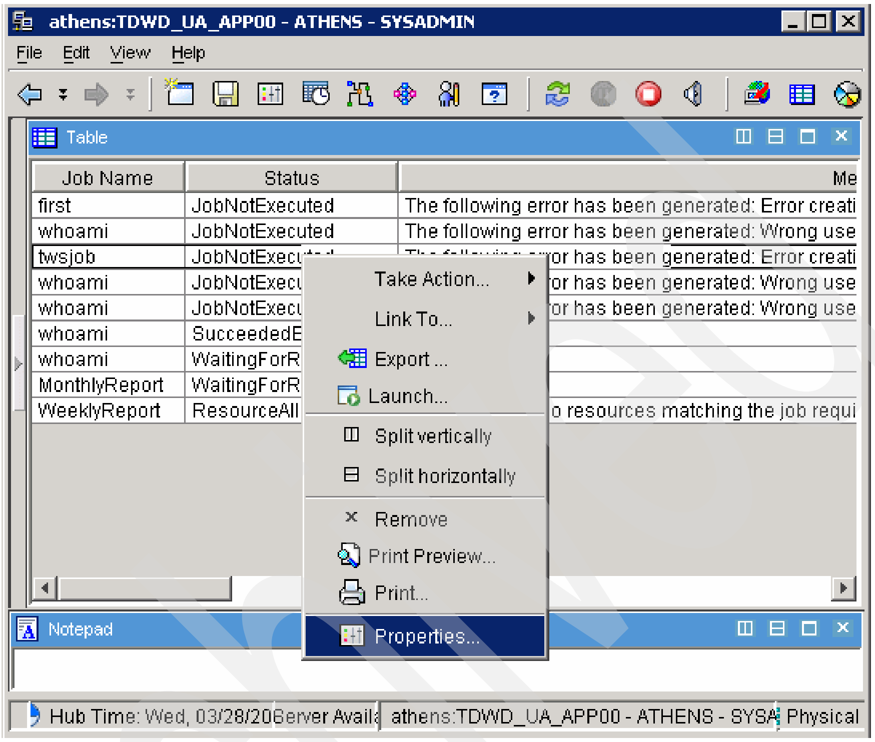
Hub Time: Wed, 03/28/20 Server Avail athens:TDWD_UA_APP00 - ATHENS - SYSA Physical

Figure 8-40 Tivoli Enterprise Portal - view without thresholds

To distinguish the way in which the data are represented, you must define *thresholds* for a particular view. Thresholds determine which data should be assigned specified severity (informational, warning, and critical).

To define thresholds do the following steps:

1. Right-click anywhere in the view and select **Properties** (Figure 8-41).



Job Name	Status	Me
first	JobNotExecuted	The following error has been generated: Error creati
whoami	JobNotExecuted	The following error has been generated: Wrong use
twsjob	JobNotExecut	The followi or has been generated: Error creati
whoami	JobNotExecu	or has been generated: Wrong use
whoami	JobNotExecu	or has been generated: Wrong use
whoami	SucceededE	
whoami	WaitingForR	
MonthlyReport	WaitingForR	
WeeklyReport	ResourceAll	o resources matching the job requi

Hub Time: Wed, 03/28/20Server Avail | athens:TDWD_UA_APP00 - ATHENS - SYSA | Physical

Figure 8-41 Setting up thresholds - step 1

2. Click the **Thresholds** bookmark. Each row represents a threshold for a specified condition. Each row has more columns, and there is an AND relationship among them. This means that the threshold condition is fulfilled only if *all of* the conditions in row are true.

   The frst column determines the severity. Possible choices are informational, warning, and critical.

   Any of next columns represents one condition. There must be at least one condition set for the threshold to become active.

   In our scenario we decided to set the critical threshold on any job, that was submitted to the appropriate resource, but was not executed for any reason. We also want to set a warning threshold on jobs for which the Tivoli Dynamic Workload Broker server did not allocate any suitable resource within a defined period.

First we explain how to manipulate GUI for setting a particular condition. In the following window situation editor click in the first numbered row on the cell located below the Current State column. Two small buttons appeared in the cell. The first button allows you to specify what we use from the monitored value. The second button is an operand (equals/not equals).

If you click into any blank cell, two small buttons appear in that cell. The first button allows us to specify what we compare in the monitored value, and the second button is an operand (equals/not equals).

In each condition that we set up, we compare the substring of an incoming monitored value with the string that we explicitly specify. For instance, if we want to set a threshold on jobs that were not submitted, we search for a substring `Not` within a job status.

Now set up the two thresholds that we mentioned above:

– CRITICAL if a job is not executed - Go to the first row and in the first column leave severity set to CRITICAL. In the second column select **Value of expression ==** JobNotExecuted from the drop-down list.

– WARNING if no resources were allocated for the job - GO to the second row and in the first column switch the severity to WARNING. In the second column select **Value of expression ==** ResourceAllocation from the drop-down list.

Figure 8-42 shows the example of how the particular cells should be filled in.

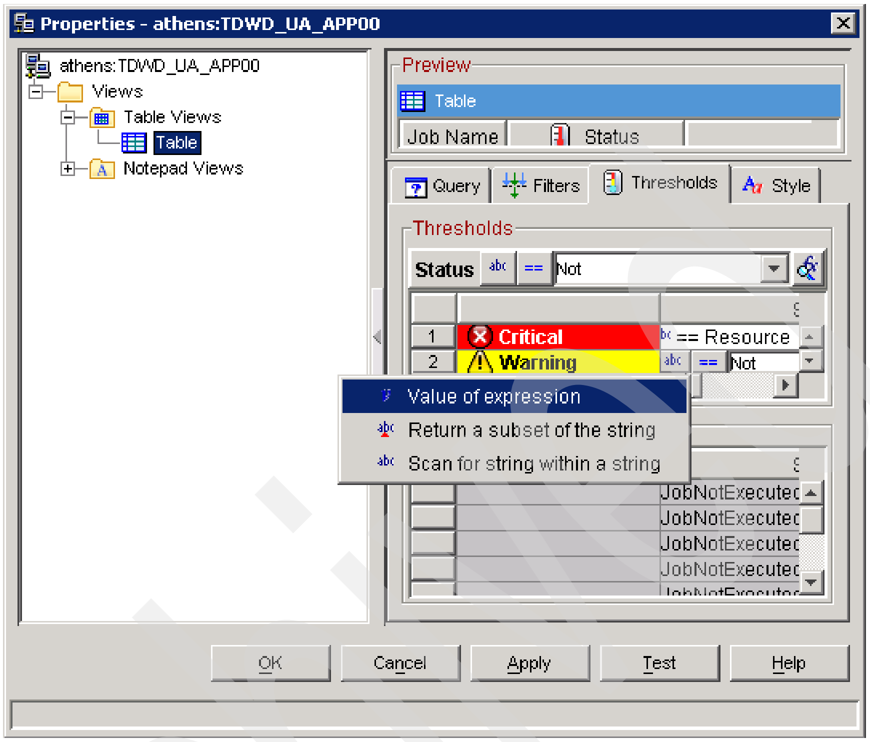
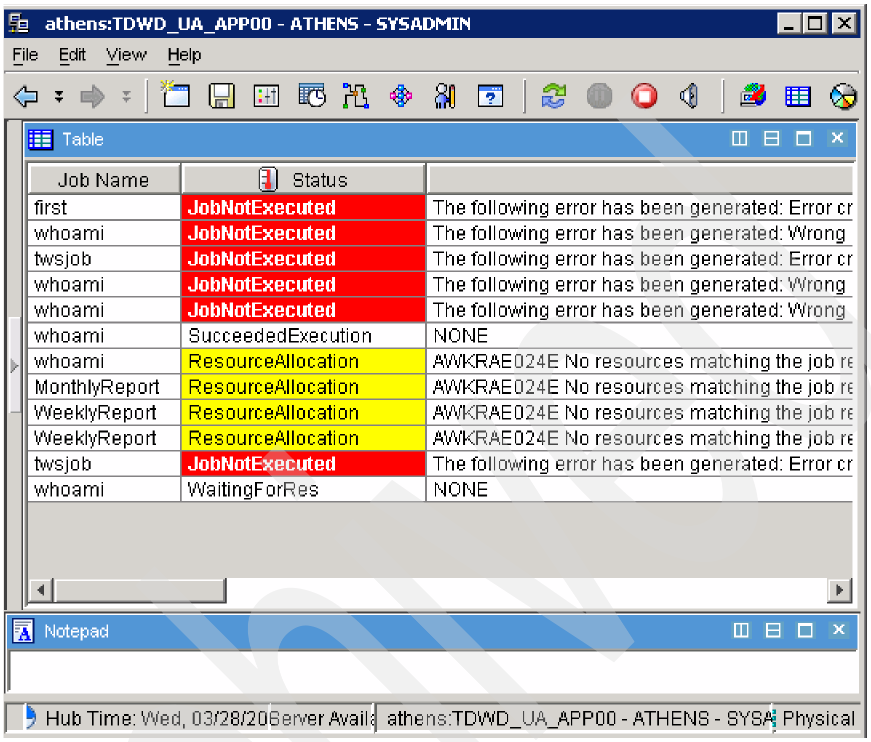


Figure 8-42 Setting up thresholds - step 2

3. Click **OK** and refresh the view. You should see that the corresponding rows have changed (Figure 8-43).



Job Name	Status	
first	JobNotExecuted	The following error has been generated: Error cr
whoami	JobNotExecuted	The following error has been generated: Wrong
twsjob	JobNotExecuted	The following error has been generated: Error cr
whoami	JobNotExecuted	The following error has been generated: Wrong
whoami	JobNotExecuted	The following error has been generated: Wrong
whoami	SucceededExecution	NONE
whoami	ResourceAllocation	AWKRAE024E No resources matching the job re
MonthlyReport	ResourceAllocation	AWKRAE024E No resources matching the job re
WeeklyReport	ResourceAllocation	AWKRAE024E No resources matching the job re
WeeklyReport	ResourceAllocation	AWKRAE024E No resources matching the job re
twsjob	JobNotExecuted	The following error has been generated: Error cr
whoami	WaitingForRes	NONE

Hub Time: Wed, 03/28/20Server Avail athens:TDWD_UA_APP00 - ATHENS - SYSA Physical

Figure 8-43 Tivoli Enterprise Portal - view with thresholds

Important: The steps in this section describe how a particular view was customized to represent monitored data. This customization has nothing to do with Tivoli Monitoring *situations* at all. Customizing a view gives you only a possibility of how to distinguish data for operators. But unless you define a situation (even with the same conditions that you have used for threshold setting in a view), you cannot see the problems from the global perspective. You must have the particular view put in your current workspace if you want to be aware of its events. Furthermore, setting a threshold in a view does not allow you to take an automatic corrective action. The steps of how to set up a situation are described in 8.4.10, “Creating situations” on page 355. The complete detailed step-by-step instructions, including snapshots, are included in “Monitoring of DB2 availability on Windows” on page 359.

8.4.10 Creating situations

In this section we explain how to create a situation that is visible from the enterprise perspective.

We implement a simple scenario that sends an SMS to a responsible system administrator if a job generating monthly financial report did not execute.

First we create the situation (described in this section) and then we define the automatic corrective action, which will send an SMS (described in 8.4.11, "Setting up automatic corrective action" on page 357).

Go to the Tivoli Enterprise Portal window and do the following steps:

1. In the Navigator pane, go to the Universal Agent under the monitored machine (Tivoli Dynamic Workload Broker server). Expand the **Universal Agent** branch and select the application name that we defined in previous steps. In our case the name is <*hostname*>:TDWB00. Right-click the application and select **Situations**.
2. The situation editor appears. Click **Create new situation** in the upper left corner. In the window that pops up, fill in the situation name `Monthly_Report_Not_Executed`, and from the drop-down list select **Universal Agent**. Click **OK**.

   We want to create situation that will be fired if these conditions are met:

   – Name of the monitored job is MonthlyReport
   – Status of the monitored job is JobNotExecuted

   First we need to select attributes that will be used for evaluating the situation formula (when to fire a situation).
3. In the left pane select the source attribute group. In the left pane click **JOB** and in the right pane select the attributes **Job Name** and **Status**.
4. Set the following evaluation criteria:

   – Job name: **Scan for substring within string ==** MonthlyReport

     Type directly into the cell.

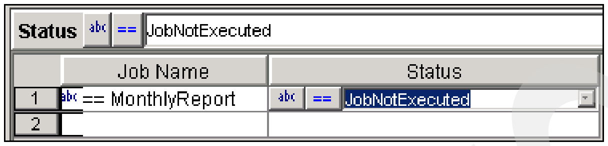
   – Status: **Scan for substring within string ==** JobNotExecuted

     Select the value from the drop-down list.

Figure 8-44 shows the situation formula.

5. Optionally, adjust the sampling interval. This interval specifies how often the Windows OS agent will check the service status and send the results to the Tivoli Monitoring server.

   The value of the sampling interval can be any that fits your needs. We use 30 seconds in our scenario, but the recommended sampling interval is usually a bit longer (from 2 to 5 minutes). Too short of a sampling interval produces unnecessary load and too long of an interval can discover a possible failure too late.


Status abc == JobNotExecuted

	Job Name	Status
1	abc == MonthlyReport	abc == JobNotExecuted
2		

Figure 8-44 Situation formula for monitoring of particular job state

> **Important:** The job states in the Tivoli Dynamic Workload Broker log file are defined using enumerations (numbers paired to the string representations). There is a known problem in the current releases of Tivoli Dynamic Workload Broker V 1.1 and Tivoli Monitoring V 6.1 that does not allow us to save the situation formula shown in Figure 8-44. This problem will be fixed in a later releases.

You have done all of the necessary steps for creating a situation that will be fired when a Tivoli Dynamic Workload Broker job with the name MonthlyReport will end with status JobNotExecuted or FailedExecution.

For more detailed step-by-step instructions (including snapshots), see a similar scenario describing monitoring of the DB2 availability in “Monitoring of DB2 availability on Windows” on page 359.

8.4.11 Setting up automatic corrective action

In this section we describe how to configure Tivoli Monitoring so that it launches a local script on the Tivoli Monitoring server, if a specified situation occurs.

Important: We assume that some *alerting script* responsible for sending SMS is located in the local file system. The alerting script is not included in this scenario, nor is it shipped with Tivoli Monitoring. This is just an example of how to link the situation with an alerting script. Depending on your installation, you should adjust the path to the script responsible for alerting.

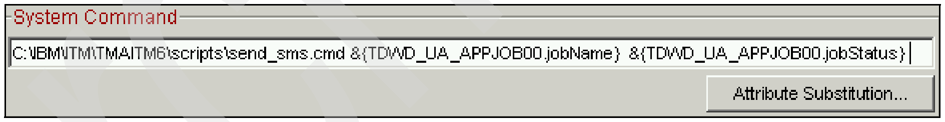
1. In the Navigator pane, go to the Universal Agent under the monitored machine (Tivoli Dynamic Workload Broker server). Expand the **Universal Agent** branch and select the application name for which we have defined the situation. The pplication name looks like *<hostname>*:TDWB00.

   Right-click the application and select **Situations**.

2. In the next window navigate in the left pane to the **Monthly_Report_Not_Executed** situation. Select it and in the right pane click **Take action**.

   Type `C:\IBM\ITM\TMAITM6\scripts\send_sms.cmd` into the text box System Command.

   Figure 8-45 shows the entire command syntax. We call the script and feed it with meaningful data.



```
C:\IBM\ITM\TMAITM6\scripts\send_sms.cmd &{TDWD_UA_APPJOB00.jobName}  &{TDWD_UA_APPJOB00.jobStatus}
```

Figure 8-45 Script for corrective action with arguments

3. Make sure that you have Execute the Action at the Managing System (TEMS) selected. Confirm by clicking **OK**.

Now you have completed the sample extension of the out-of-box integration of Tivoli Dynamic Workload Broker with Tivoli Monitoring. So far you have:

- Configured the Tivoli Dynamic Workload Broker to log to the file.
- Configured the Tivoli Monitoring Universal Agent to parse this log file.
- Configured the view in the Tivoli Enterprise Portal so that gathered data are displayed in the Tivoli Enterprise Portal.
- Set up thresholds so that gathered data are distinguished by severities.

- Created a situation to be able to see important job states from the enterprise perspective.
- Created the corrective action that is fired when a situation occurs (sending of SMS when important jobs fail).

In the following sections we describe how can we use Tivoli Monitoring to check the health of Tivoli Dynamic Workload Broker components.

8.4.12 Advanced monitoring of Tivoli Dynamic Workload Broker

The out-of-box Tivoli Dynamic Workload Broker integration with Tivoli Monitoring provides monitoring of possible job states that are processed by the Tivoli Dynamic Workload Broker. You can monitor which jobs were cancelled, which did not find their necessary resources, which of them have failed, and so on.

Out-of-box integration of the Tivoli Dynamic Workload Broker and Tivoli Monitoring does not provide monitoring of the health of Tivoli Dynamic Workload Broker components themselves.

In this section we describe how to extend the out-of-box provided monitoring of Tivoli Dynamic Workload Broker. Together with the out-of-box monitoring, we use following monitoring techniques:

- Monitoring of Windows services using a Windows OS agent
- Using Universal Agent for launching custom scripts
- Parsing content of a log file using Universal Agent (out-of-box integration described in 8.4.4, “Integration steps” on page 330)

We use these techniques to demonstrate how the Tivoli Dynamic Workload Broker components can be monitored. In our scenarios we show how to monitor:

- DB2 availability on Windows
- Tivoli Dynamic Workload Broker Agent availability on Windows
- Availability of the ITDWB enterprise application running in the WebSphere Application Server on Windows

Note: All of the scenarios included in this chapter were performed in a Windows environment. However, the steps on UNIX/Linux platforms are similar. The different default paths and commands are listed in 8.4.13, “Default values and file locations” on page 391.

Monitoring of DB2 availability on Windows

In this scenario we demonstrate how to monitor the availability of DB2. The Tivoli Dynamic Workload Broker server uses DB2 as its persistent storage and thus DB2 is a very important component.

As a simple descriptive example we have chosen a monitoring of Windows DB2 services. We use the standard Windows operating system agent for periodical checking of DB2 service availability. If the service goes down, the Tivoli Monitoring server fires a situation and launches a corrective action that restarts the DB2 service.

Note: Before you start monitoring the DB2 used by the Tivoli Dynamic Workload Broker server, you must deploy an Tivoli Monitoring Windows OS agent to the system where DB2 is installed. This is not necessary if the Windows OS agent is already installed on the target system.

In following steps we explain how to:

- View the default monitoring of a Windows service.
- Create a situation on *service down* state.
- Create a corrective action *restart service.*

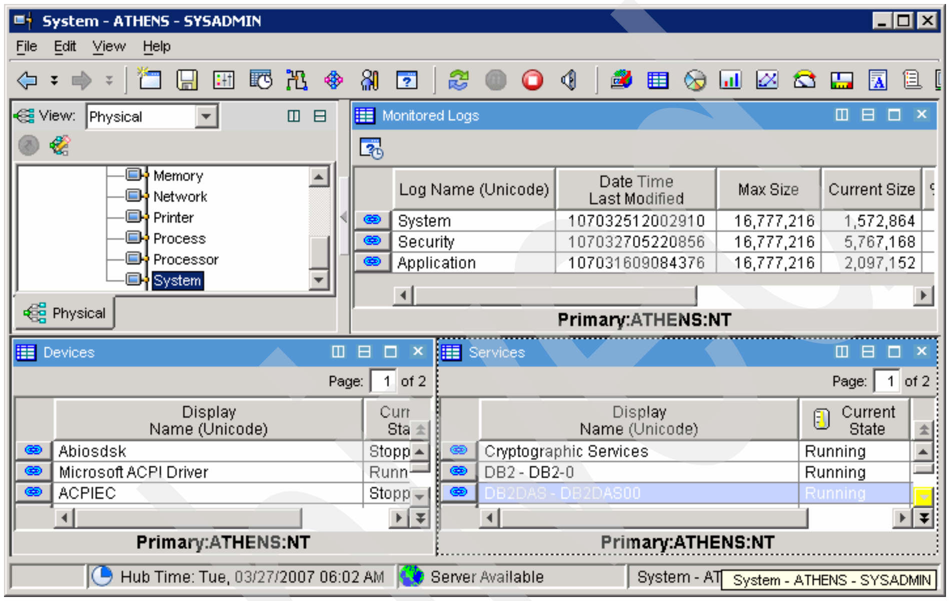
Default monitoring of Windows services

The Tivoli Monitoring Windows OS agent can monitor many system values, such as CPU utilization, memory usage, available disk space, and so on.

The important value for us is monitoring of the state of particular *service*. We use a Tivoli Monitoring OS Agent to periodically check the states of DB2 services. In this example we monitor all of the DB2 services that are configured to start automatically during Windows startup. We do not monitor the DB2 services that are started manually.

The first step is to ensure that the Windows OS agent is able to see the service status. This option is available by default after the Windows OS agent is deployed onto the target system.

In the Navigator pane expand the branch of the monitored system (server where DB2 is running and the Windows OS agent is installed). After that expand the **Windows OS** and select **System**. Look at the right-bottom pane. The window should display states of Windows services. Scroll down if necessary and search for DB2-related services. See the Figure 8-46.



Log Name (Unicode)	Date Time Last Modified	Max Size	Current Size
System	1070325120029l0	16,777,216	1,572,864
Security	107032705220856	16,777,216	5,767,168
Application	107031609084376	16,777,216	2,097,152

Primary:ATHENS:NT

Display Name (Unicode)	Curr Sta
Abiosdsk	Stopp
Microsoft ACPI Driver	Runn
ACPIEC	Stopp

Primary:ATHENS:NT

Display Name (Unicode)	Current State
Cryptographic Services	Running
DB2 - DB2-0	Running
DB2DAS - DB2DAS00	Running

Primary:ATHENS:NT

Figure 8-46 Default Windows services monitoring

This shows you that the Windows OS agent monitors the services properly.

Creating a situation for unavailable DB2 service

In this section we describe how to create a *situation* that will be fired when one of the DB2 services crashes or stops.

Do the following steps:

1. In the Navigator pane, go to the Windows OS under the monitored machine (our DB2 server). Right-click the **Windows OS** and select **Situations** (Figure 8-47).

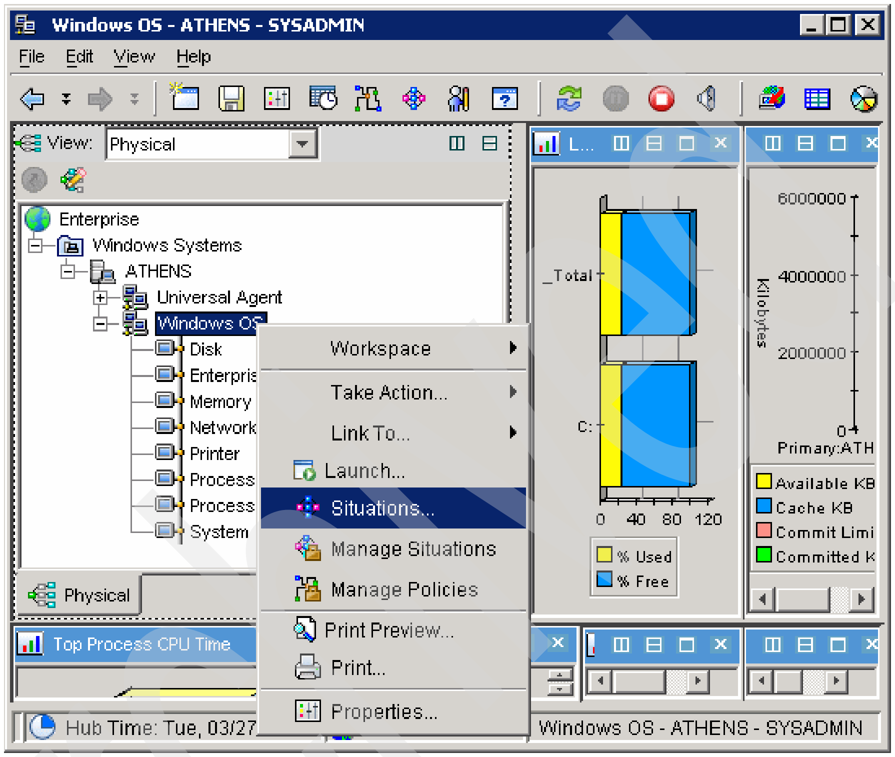
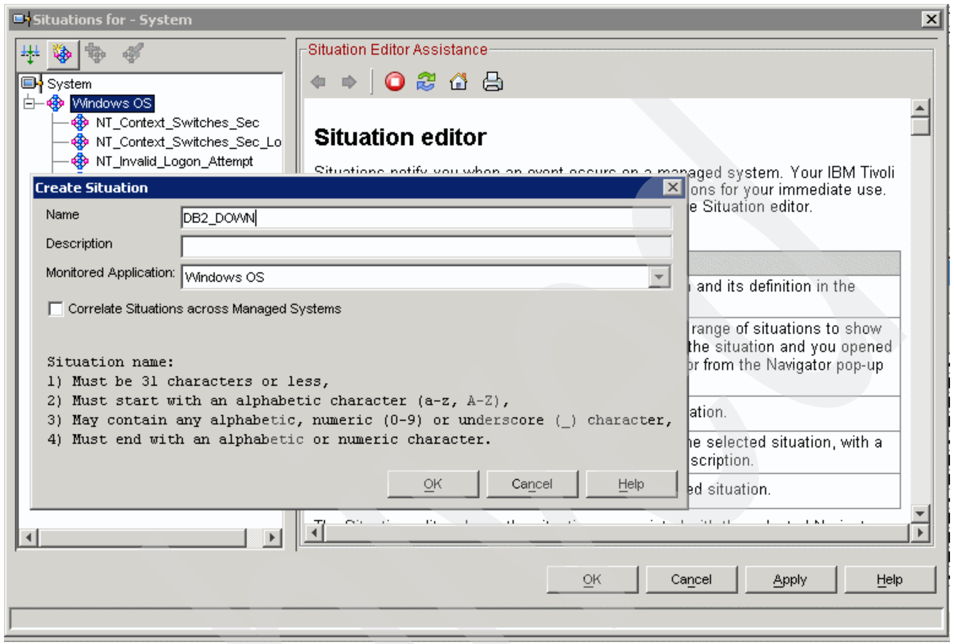


Figure 8-47 Creating a situation - step 1

2. A situation editor appears. Click **Create new situation** in the upper left corner. In the window that popped up, fill in the situation name `DB2_DOWN` and from the drop-down list select **Windows OS** (Figure 8-48).



Create Situation

Name: DB2_DOWN

Description:

Monitored Application: Windows OS

☐ Correlate Situations across Managed Systems

```
Situation name:
1) Must be 31 characters or less,
2) Must start with an alphabetic character (a-z, A-Z),
3) May contain any alphabetic, numeric (0-9) or underscore (_) character,
4) Must end with an alphabetic or numeric character.
```

OK Cancel Help

Figure 8-48 Creating a situation - step 2

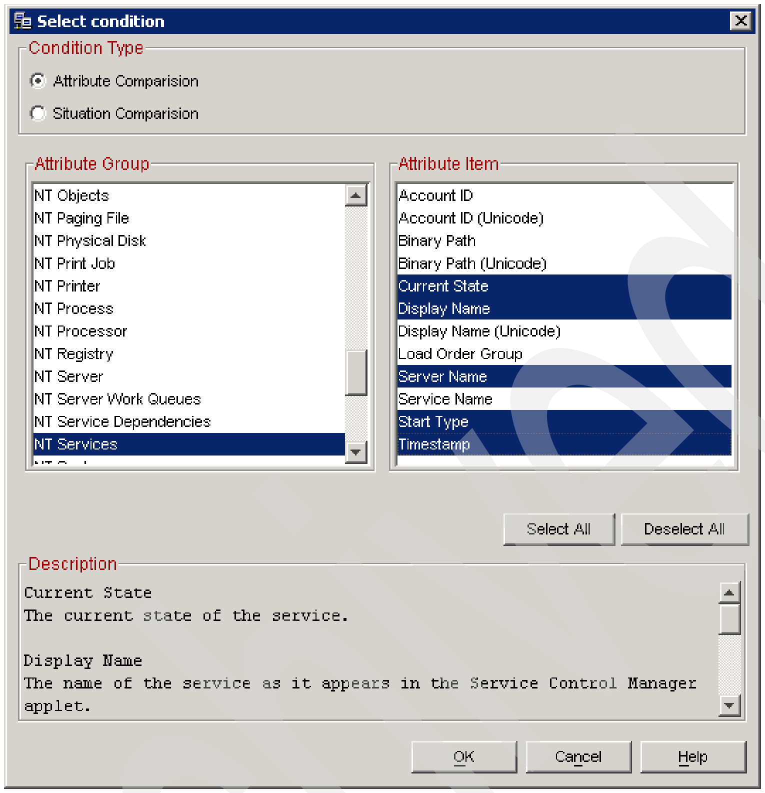
3. Now select the attributes that we will use for evaluating a situation. We want to fire a *critical* situation when following conditions are met:

   – Server name of the monitored system is our"host name
   – Status of service is different from *running*
   – Service startup type is *automatic*
   – Name of monitored service contains string *DB2*

   In the left pane click **NT_Services**. In the right pane choose following attributes:

   – Current state
   – Display name (In this scenario you can use *service name* as well.)
   – Server name
   – Start type

Figure 8-49 shows the selected attributes.



Condition Type: [✓ Attribute Comparision] Situation Comparision

Attribute Group: NT Objects, NT Paging File, NT Physical Disk, NT Print Job, NT Printer, NT Process, NT Processor, NT Registry, NT Server, NT Server Work Queues, NT Service Dependencies, [✓ NT Services]

Attribute Item: Account ID, Account ID (Unicode), Binary Path, Binary Path (Unicode), [✓ Current State], [✓ Display Name], Display Name (Unicode), Load Order Group, [✓ Server Name], Service Name, [✓ Start Type], [✓ Timestamp]

```
Current State
The current state of the service.

Display Name
The name of the service as it appears in the Service Control Manager
applet.
```

Figure 8-49 Creating a situation - step 3

4. The ext window contains a *situation formula editor*. In this window you define the *situation formula*, which describes when the situation should be fired. The situation formula consists of one or more evaluation criteria.

In the following window situation editor click in the first numbered row on the cell located below the Current State column. Two small buttons appear in the cell. The first button allows you to specify what we compare in the monitored value, and the second button is an operand (equals/not equals). See Figure 8-50.

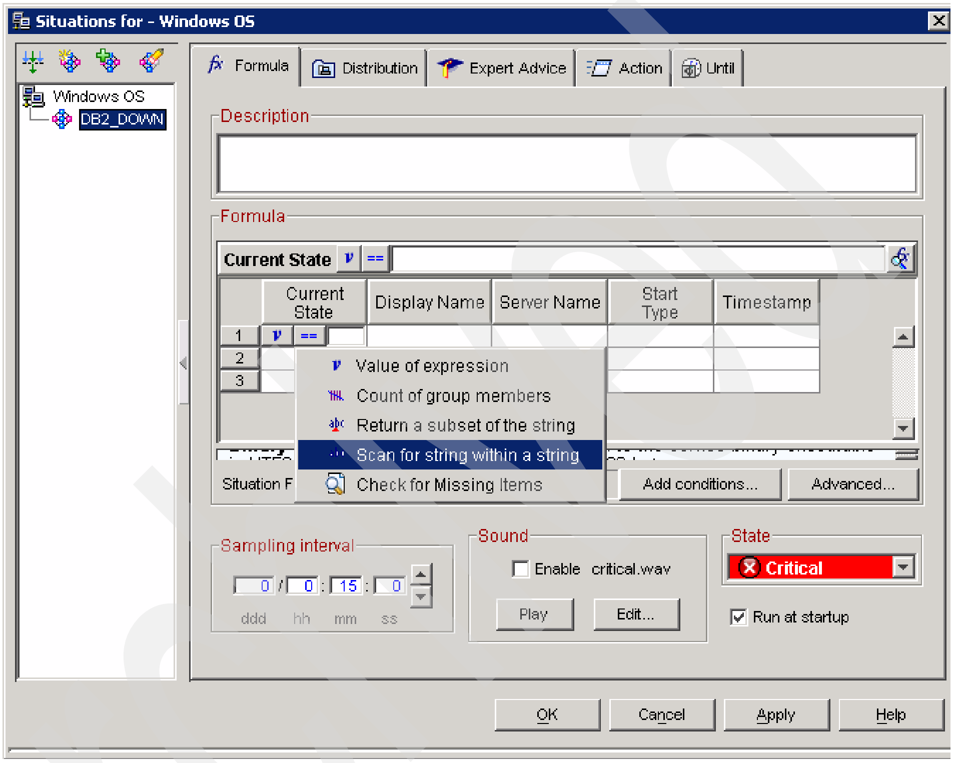


Figure 8-50 Creating a situation - step 4

5. As an action, select **Scan for string within a string**. Then select the**!=** (not equals) attribute. Finally, fill in the value *Running*.

   In this step you have specified one evaluation criteria, which instruct the Tivoli Monitoring server to fire a situation when a data sample about any not-running service arrives.

   This is still not enough. If we specified only this one evaluation criteria a situation would be fired on a crashed/stopped service.

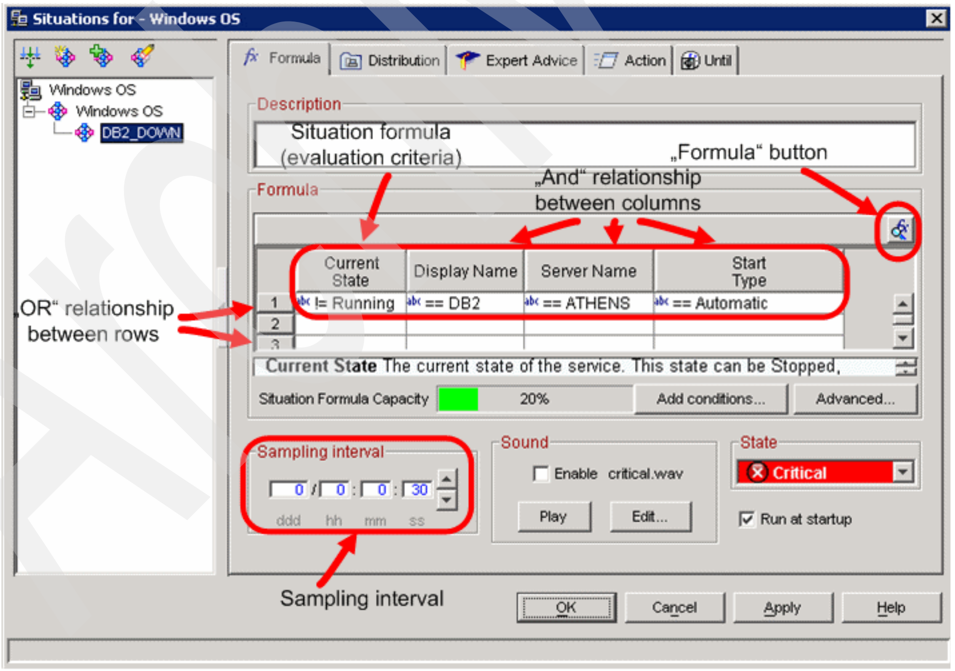
We must specify a complete situation formula for a successful situation evaluation:

– Status of service is different from *running* (already done in previous step)
– Name of monitored service contains string DB2
– Server name of the monitored system is our host name
– Service startup type is **`automatic`**

See Figure 8-51. Fill in the cells as shown in the example. For all columns use the Scan for string within a string function and == (equals) operand. The only case when we select the != (not equals) operand is the Current state column because we are searching for occurrences other than *running*.

6. Now adjust the sampling interval. This interval specifies how often the Windows OS agent checks the service status and sends the results to the Tivoli Monitoring server.

   The value of the sampling interval can be any that fits your needs. We have used 30 seconds in our scenario, but the recommended sampling interval is usually a bit longer (from 2 to 5 minutes). A too-short sampling interval produces unnecessary load, and a too-long interval can discover a possible failure too late.



	Current State	Display Name	Server Name	Start Type
1	!= Running	== DB2	== ATHENS	== Automatic
2				
3				

Current State The current state of the service. This state can be Stopped,

Situation Formula Capacity 🟩 20%

Sampling interval: 0 / 0 : 0 : 30 (ddd hh mm ss)

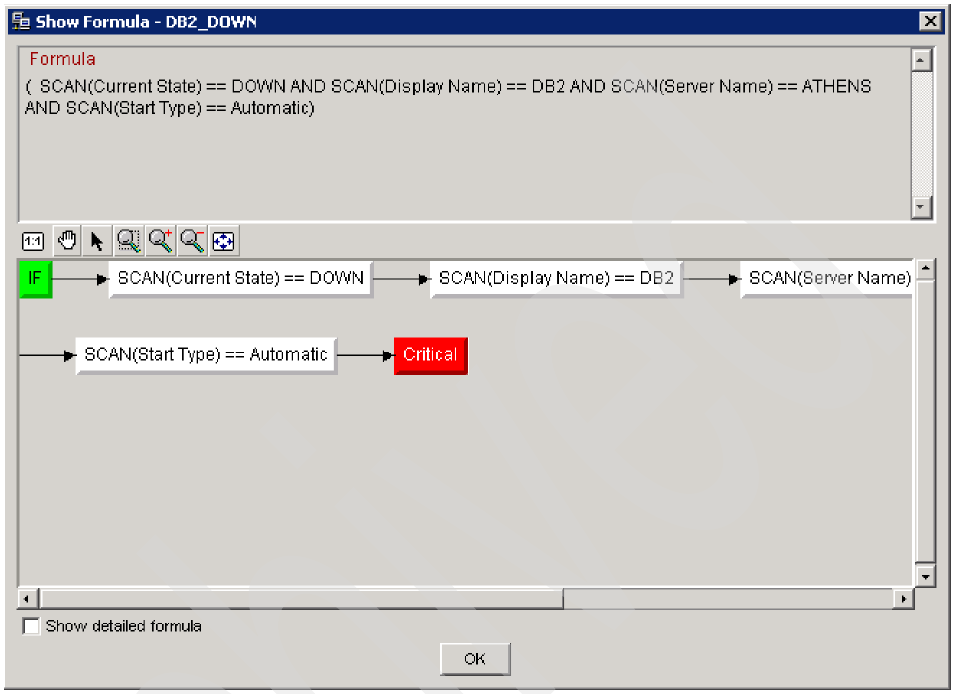
Sound: ☐ Enable critical.wav

State: Critical

☑ Run at startup

Figure 8-51 Creating a situation - step 5

7. Click the **Formula** button to see our formula. Note that we shrank the output to one window for better visibility. In a real window, you need to move the slider to see the entire formula (Figure 8-52).


Show Formula - DB2_DOWN
Formula

(SCAN(Current State) == DOWN AND SCAN(Display Name) == DB2 AND SCAN(Server Name) == ATHENS AND SCAN(Start Type) == Automatic)

IF
SCAN(Current State) == DOWN
SCAN(Display Name) == DB2
SCAN(Server Name)
SCAN(Start Type) == Automatic
Critical
Show detailed formula
OK

Figure 8-52 Creating a situation - step 6

8. Close this window. Click **OK** and you are done with defining the situation that monitors availability of DB2 services on Windows.

Testing the monitoring settings

In this section we describe a simple testing scenario. We test whether the situation will fire when one DB2 service crashes or stops.

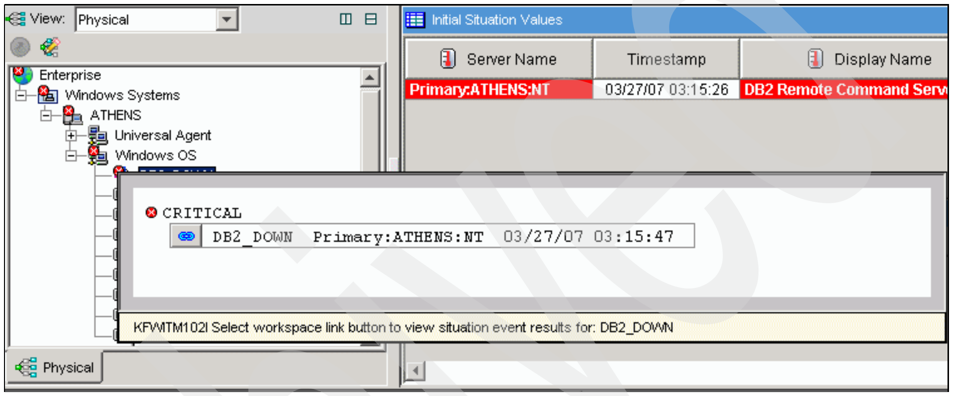
Note: We strongly recommend doing the following steps *only* in a *testing* environment. Do not stop any service in a production environment unless you have a maintenance window arranged.

Do the following steps:

1. Stop one of the DB2 services, for instance, the DB2 Remote Command Server.

Stop this service either via Windows Service Control Manager or using the command line.

2. Switch to the Tivoli Enterprise Portal and observe the Navigator pane. Wait for the same interval that you specified when defining the situation. Then the view should change and error markings should be added into the Navigator tree. If you point the mouse directly over the error marking, a hoover help will pop up with more detailed information. Depending on your view configuration and your position in the navigator pane, you should see output similar to Figure 8-53.



Server Name	Timestamp	Display Name
Primary:ATHENS:NT	03/27/07 03:15:26	DB2 Remote Command Serv

```
CRITICAL
DB2_DOWN   Primary:ATHENS:NT   03/27/07 03:15:47
```

KFWITM102I Select workspace link button to view situation event results for: DB2_DOWN

Figure 8-53 Situation - DB2 service down with hoover help

We have successfully tested the monitoring of DB2 services.

Setting up a corrective action

In this section we describe how to set up a corrective action when DB2 service crashes or stops. We set up a corrective action that restarts the crashed service automatically.

Do the following steps:

1. In the Navigator pane point to the server where we previously created the DB2_DOWN situation (described in “Creating a situation for unavailable DB2 service” on page 360). Expand the branch and navigate to Windows OS. Right-click the icon and select **Situations** (Figure 8-54).

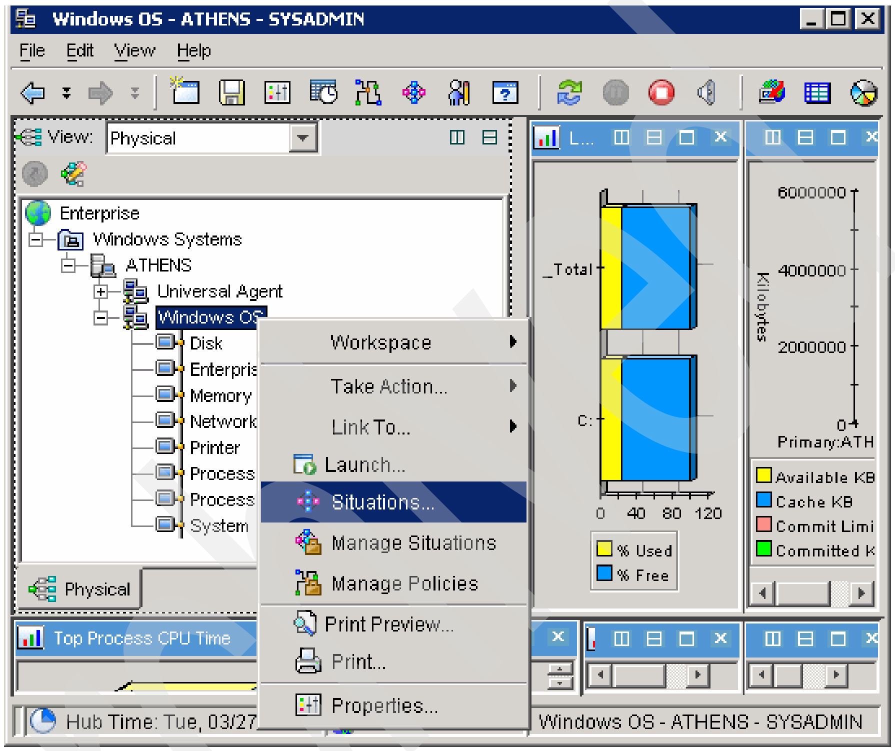
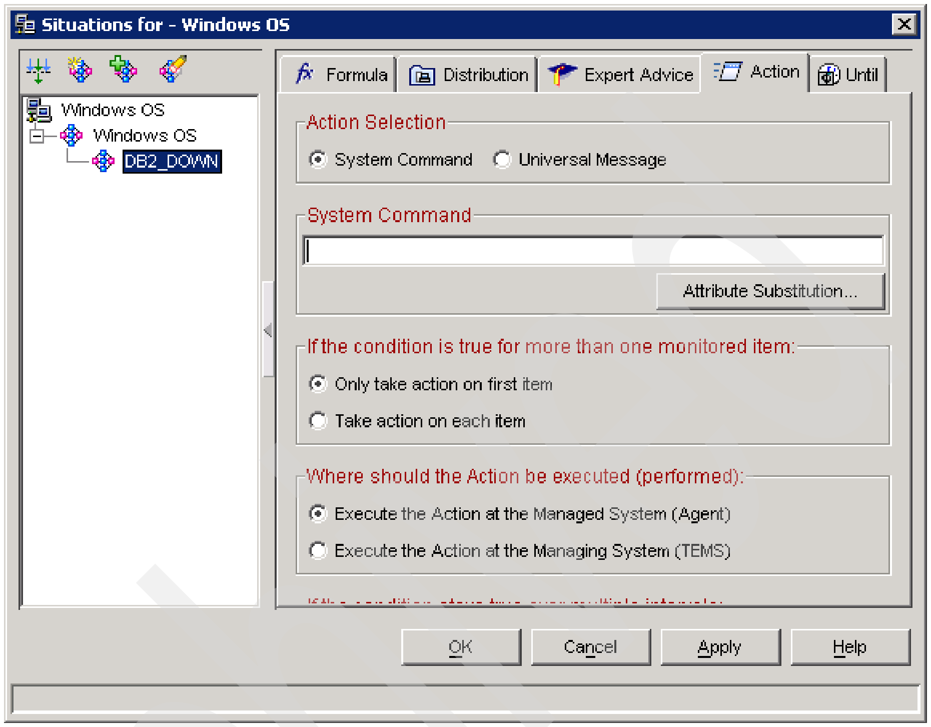


Figure 8-54 Adding an action to situation- step 1

2. In the following window navigate in the left pane to the **DB2_DOWN** situation. Select it and in the right pane select **Take action** (Figure 8-55).



Action Selection
(•) System Command ( ) Universal Message

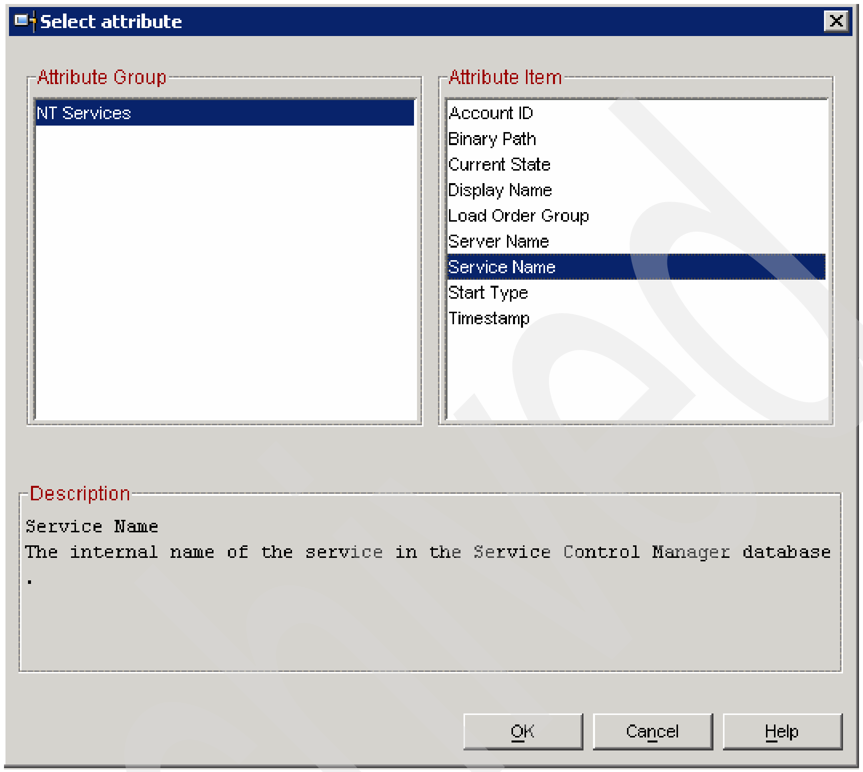
System Command
Attribute Substitution...

If the condition is true for more than one monitored item:
(•) Only take action on first item
( ) Take action on each item

Where should the Action be executed (performed):
(•) Execute the Action at the Managed System (Agent)
( ) Execute the Action at the Managing System (TEMS)

Figure 8-55 Adding an action to situation - step 2

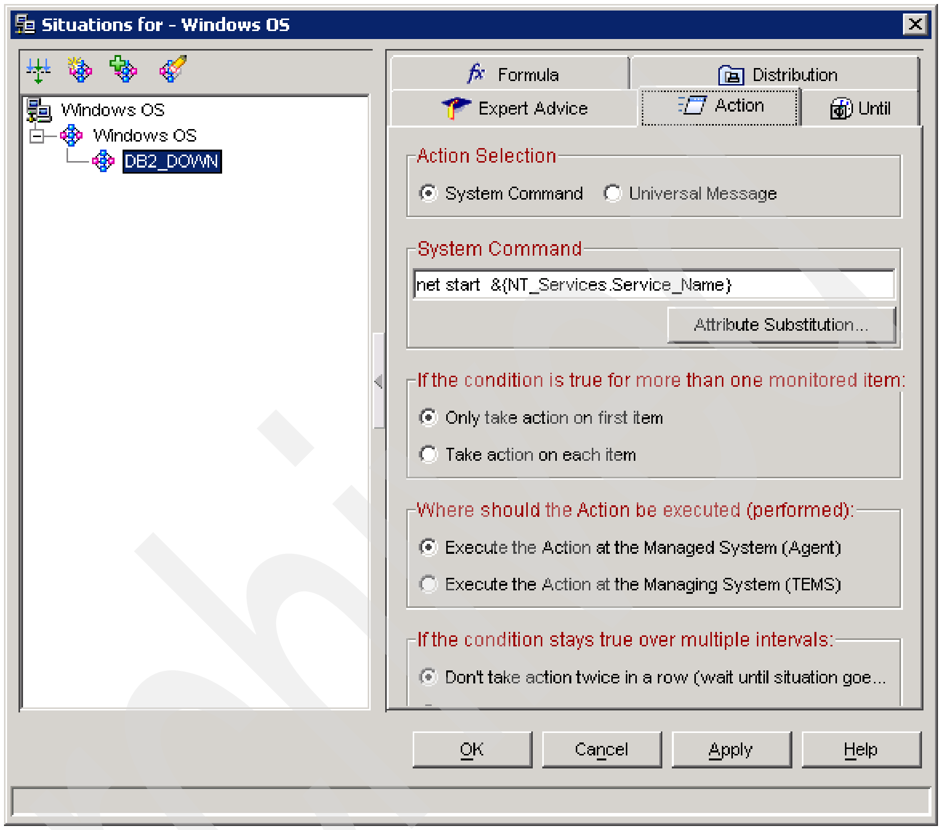
3. Now type `net start` into the System Command text box. Click **Attribute Substitution**. In the next window select **Service Name** (Figure 8-56).



```
Service Name
The internal name of the service in the Service Control Manager database
.
```

Figure 8-56 Adding an action to situation - step 3

4. Click **OK** and look at the result in the System command text box. The command was extended by a variable representing the name of the service that fired the situation (Figure 8-57).


Situations for - Windows OS

Windows OS
- Windows OS
  - DB2_DOWN

Formula | Distribution | Expert Advice | Action | Until

Action Selection

[✓ System Command] Universal Message

System Command

```
net start  &{NT_Services.Service_Name}
```

Attribute Substitution...

If the condition is true for more than one monitored item:

[✓ Only take action on first item]

Take action on each item

Where should the Action be executed (performed):

[✓ Execute the Action at the Managed System (Agent)]

Execute the Action at the Managing System (TEMS)

If the condition stays true over multiple intervals:

[✓ Don't take action twice in a row (wait until situation goe...]

OK | Cancel | Apply | Help

Figure 8-57   Adding an action to situation - step 4

5. Make sure that you have Execute Action at the Managed system (Agent) selected. Confirm by clicking **OK**.

   This is the last step of setting up the corrective action.

Testing the corrective action

To test whether the service gets restarted after it has crashed (or was stopped), do the following steps. They are in fact the same as in testing the situation.

Note: We strongly recommend doing the following steps *only* in a *testing* environment. Do not stop any service in a production environment unless you have a maintenance window arranged.

1. Stop one of DB2 service, for instance, the DB2 Remote Command Server.
2. Stop this service either via Windows Service Control Manager or using a command line.
3. Wait for the same interval that you specified when defining the situation. After that refresh the Service Control Manager. The service should be running again.

Monitoring of Tivoli Dynamic Workload Broker Agent on Windows

In this section we describe how to monitor the availability of the Tivoli Dynamic Workload Broker agent.

Each Tivoli Dynamic Workload Broker agent runs on top of the Tivoli Common Agent. In fact, the main run time registered as a Windows service is the Common Agent, and the Tivoli Dynamic Workload Broker agent runs only as a *subagent* of the Common Agent. For more information about this topic see 2.4, “Common Agent Services” on page 38.

On Windows the Common Agent is registered as a service. In our scenario we use the Tivoli Monitoring Windows OS agent to monitor the Common Agent. The Windows OS agent can monitor many system values, such as CPU utilization, memory usage, available disk space, and so on. For our purposes we configure the Windows OS agent to periodically check the state of a Windows service for Common Agent.

Setting up the monitoring of the Tivoli Dynamic Workload Broker Agent is similar to monitoring DB2, as described in “Monitoring of DB2 availability on Windows” on page 359. The steps for monitoring the Tivoli Dynamic Workload Broker agent availability are almost the same.

Note: Before you start monitoring the Tivoli Dynamic Workload Broker Agent, you must deploy an Tivoli Monitoring Windows OS agent to the target system. This is not necessary if the monitoring agent is already installed on the system where Tivoli Dynamic Workload Broker Agent runs.

We state all of the necessary steps for monitoring of a Common Agent' Windows service. We provide snapshots only for the most important part — situation formula editor. If you are interested in looking at all snapshots describing the setup of Windows service monitoring, read "Monitoring of DB2 availability on Windows" on page 359 first.

Determining the name of monitored service

In this scenario we use the exact name of the monitored service.

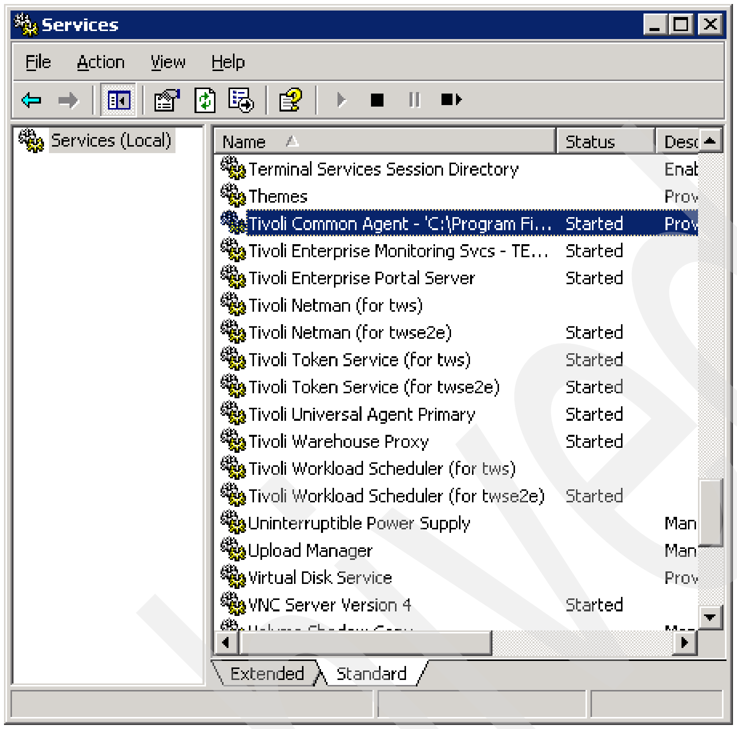
We want to create situation that will be fired when the Tivoli Dynamic Workload Broker Agent service is not running. To do this, we set a situation formula to evaluate the following criteria:

- Status of service is different from *running*
- Name of monitored service is exactly the same as name Tivoli Dynamic Workload Broker Agent service

We must determine the exact name of the Common Agent hosting the Tivoli Dynamic Workload Broker Agent. Because of that, we must first find out the name of the Common Agent service.

1. Launch the Service Control Manager (**Start** → **Settings** → **Control Panel** → **Administrative Tools** → **Services**).

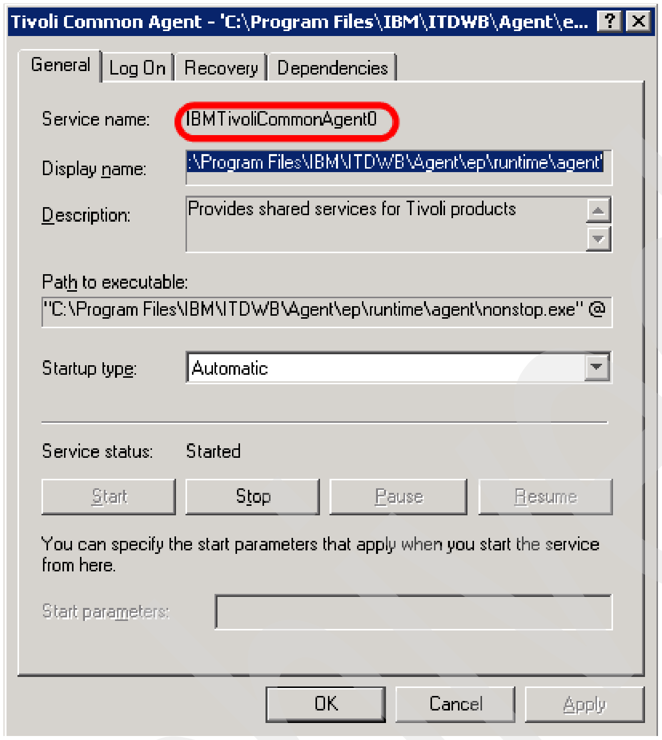
2. Within the services, select the **Tivoli Common Agent** (Figure 8-58).



Name	Status	Desc
Terminal Services Session Directory		Enat
Themes		Prov
Tivoli Common Agent - 'C:\Program Fi...	Started	Prov
Tivoli Enterprise Monitoring Svcs - TE...	Started	
Tivoli Enterprise Portal Server	Started	
Tivoli Netman (for tws)		
Tivoli Netman (for twse2e)	Started	
Tivoli Token Service (for tws)	Started	
Tivoli Token Service (for twse2e)	Started	
Tivoli Universal Agent Primary	Started	
Tivoli Warehouse Proxy	Started	
Tivoli Workload Scheduler (for tws)		
Tivoli Workload Scheduler (for twse2e)	Started	
Uninterruptible Power Supply		Man
Upload Manager		Man
Virtual Disk Service		Prov
VNC Server Version 4	Started	

Figure 8-58 Tivoli Common Agent in Service Control Manager

3. Double-click it and read the service name (Figure 8-59).


Tivoli Common Agent - 'C:\Program Files\IBM\ITDWB\Agent\e...
General
Log On
Recovery
Dependencies

Service name: IBMTivoliCommonAgent0

Display name: :\Program Files\IBM\ITDWB\Agent\ep\runtime\agent'

Description: Provides shared services for Tivoli products

Path to executable:
"C:\Program Files\IBM\ITDWB\Agent\ep\runtime\agent\nonstop.exe" @

Startup type: Automatic

Service status: Started

You can specify the start parameters that apply when you start the service from here.

Start parameters:

Figure 8-59 Tivoli Common Agent service name

We will use this service name in the situation formula editor.

Setting up the monitoring of Agent service availability

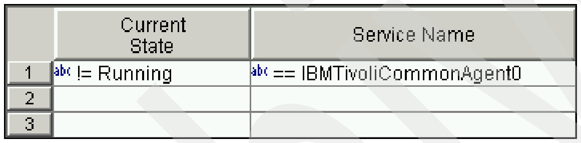
Now we describe the steps that must be taken in order to configure Tivoli Monitoring to fire a situation when a Common Agent (hosting the Tivoli Dynamic Workload Broker agent) is not running.

1. In the Navigator pane, go to Windows OS under the monitored machine (server where Common Agent runs). Right-click **Windows OS** and select **Situations**.
2. The situation editor appears. Click **Create new situation** in the upper left corner. In the window that pops up, fill in the situation name `TDWB_Agent_DOWN`, and from the drop-down list select **Windows OS**.
3. We want to create a situation that will be fired when the agent service is not running. First select **NT_Services** in the left pane and then select **Service Name** and **Current State** in the right pane.

4. In the situation formula editor set the following evaluation criteria:
   - Current status: **Scan for string within a string !=** Running
   - Service name: **Scan for string within a string ==** IBMTivoliCommonAgent0

   We determined the name of the service in "Determining the name of monitored service" on page 373. In our scenario, the service name is IBMTivoliCommonAgent0.
5. Optionally, adjust the sampling interval. This interval specifies how often the Windows OS agent will check the service status and send the results to the Tivoli Monitoring server.

   The value of the sampling interval can be any that fits your needs. We use 30 seconds in our scenario, but the recommended sampling interval is usually a bit longer (from 2 to 5 minutes). Too short of a sampling interval produces unnecessary load, and too long of an interval can discover a possible failure too late.



	Current State	Service Name
1	!= Running	== IBMTivoliCommonAgent0
2		
3		

Figure 8-60 Situation formula for monitoring of Agent service

6. Set up the correction action, if you want to. A corrective action will do the following: each time the Windows OS agent detects a crashed (or stopped) service of a Common Agent (host of Tivoli Dynamic Workload Broker Agent), it will attempt to restart it automatically.

   To set up the corrective action, click the **Action** bookmark, and into the text box System Command type `net start`. Click **Attribute Substitution** and in the window that pops up, select **Service Name**.
7. Make sure that you have **Execute Action at the Managed system (Agent)** selected. Confirm by clicking **OK**.

   You have done all of the necessary steps for creating a situation with its corrective action.

   For more detailed step-by-step instructions (including snapshots), see a similar scenario describing monitoring of the DB2 availability in "Monitoring of DB2 availability on Windows" on page 359.

Distributing monitoring of Universal Agent across environment

In the previous section we defined a situation that is fired when a Windows service registered for Common Agent stops or crashes. In this section we describe how to distribute this situation to other machines so that we can monitor multiple instances of the Common Agent across the IT environment.

To distribute the defined situation to another system:

1. In the Navigator pane go to the Windows OS where we previously defined the situation for the crashed/stopped Common Agent service. Right-click the icon.
2. In the following window navigate to the previously defined situation **TDWB_Agent_DOWN** and then select the **Distribution** bookmark.
3. In the right pane select the systems where you want to monitor the availability of the Common Agent. Be aware that only the systems that have Windows OS agent installed are visible in the list.
4. Click the left arrow in the middle of the window.
5. Click **OK**.

Now you have modified the situation so that it fires whenever Common Agent fails on any of selected systems.

Monitoring the ITDWB Enterprise Application

In “Monitoring of DB2 availability on Windows” on page 359 and “Monitoring of Tivoli Dynamic Workload Broker Agent on Windows” on page 372 we described how to use a Windows OS agent to monitor a specified Windows service availability.

A similar approach with some differences can be used for monitoring on UNIX and Linux platforms. Instead of monitoring services, we would focus on monitoring processes.

In this section we show a different technique. We describe how to use the Tivoli Monitoring Universal Agent to run a script that periodically checks the status of the Tivoli Dynamic Workload Broker enterprise application running in the WebSphere Application Server.

Custom monitoring scripts for WebSphere Application Server

The WebSphere Application Server provides by default two administrative interfaces:

- Web administrative interface called Administrative console. This is an enterprise application that offers a Web interface for managing the instance of the WebSphere Application Server, enterprise applications running in the WebSphere Application Server, and so on.
- Command line tool called `wsadmin`. Provides similar functionality as the Administrative Console, but all of the commands are issued from the command line and thus can be used in scripts.

The **wsadmin** command serves for managing of the WebSphere Application Server from teh command line. It reads instructions either from command line or from a supplied file. When using simple commands, it is sufficient to issue the **wsadmin** command with all parameters supplied directly in the command line. For more complex actions or queries, instructions must be passed within a file.

There are two possible languages that can be used for scripting with the **wsadmin** command:

- Jacl - a Java implementation of the TCL language
- Jython - a Java implementation of the Python language

For our simple scenario we chose jacl language. In Example 8-4 we show the script that is passed to the **wsadmin** command. We use this script in our scenario. The script checks whether the ITDWB enterprise application (the Tivoli Dynamic Workload Broker server) is running in the WebSphere Application Server.

Example 8-4 Jacl script for checking enterprise application status

```
set result_string [$AdminControl completeObjectName \
type=Application,name=ITDWB,*]
set TDWB_STATUS [string first ITDWB $result_string]

if {$TDWB_STATUS == -1} {
 puts "TDWB is NOT running!!"
} else {
 puts "TDWB is running."
}
```

Note: The first line ends with a backslash (\). This is a typical UNIX convention instructing the interpreter that the line is not completed and continues on the following line.

It is not necessary to know the jacl syntax or wsadmin instructions at this time. The logic,contained in the file is simple:

- The output of the first command is either empty (the requested enterprise application is not running) or contains a message including the application name.
- The rest of the file is just an evaluation of whether the first command exited with an empty message. We try to find a substring ITDWB in the output of the previous command. If we find this string, the script exits with the result that TDWB is running. In the opposite case, the script exits with the result that TDWB is not running.

Save this file into file C:\IBM\ITM\TMAITM6\scripts\websphere_check_tdwb.jacl.

For the UNIX default path see Table 8-4 on page 392.

Calling the instructions for wsadmin script from command line

To use the wsadmin with the script file you must do the following:

1. Source the environment used by the WebSphere Application Server.
2. Run the **wsadmin** command located in the bin subdirectory of the WebSphere Application Server installation directory with the following arguments:

   wsadmin -f *name_of_script_file*

> **Note:** Scripts launched by Tivoli Monitoring Universal Agent can be supplied with separated *envfile*, which sources the necessary environment. In our scenario we do not use Universal Agent envfile, we set the environment variables directly in the script.

Example 8-5 shows the complete script on Windows.

Example 8-5 Windows script used to call wsadmin interface

```
@echo off
SET SystemRoot=C:\WINDOWS
c:\progra~1\ibm\websphere\appserver\bin\wsadmin -f c:\ibm\itm\tmaitm6\s
cripts\websphere_check_tdwb.jacl
```

This script contains the minimal necessary set of environmental variables used for the **wsadmin** command for WebSphere Application Server 6.0 on Windows.

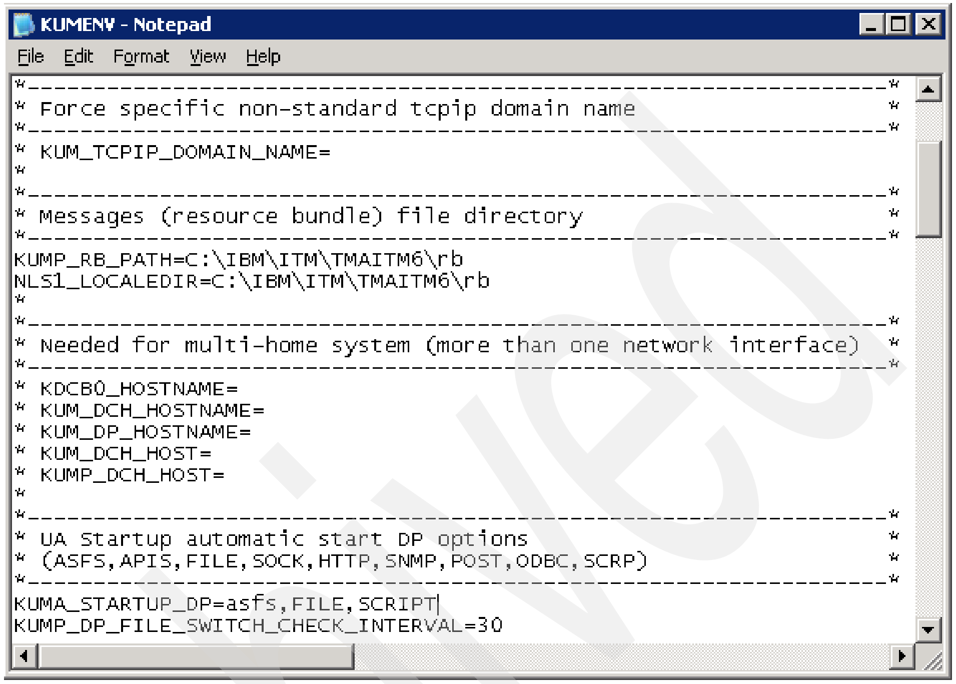
Save the content into file C:\IBM\ITM\TMAITM6\scripts\websphere_start_tdwb.cmd.

Now we have to configure the Tivoli Monitoring Universal Agent to periodically launch this file and evaluate its output.

Configuring Universal Agent to use script data provider

Even if the Tivoli Monitoring Universal Agent should accept script files as its data provider by default, we experienced different behavior in our scenarios. Unless we explicitly specified a “SCRIPT” data provider in the configuration of Universal Agent, we did not see any input from the Tivoli Dynamic Workload Broker server.

For setting the script data provider as a data source for a particular Universal Agent (the Universal Agent, which runs on the same machine as the DB2 server), see 8.4.6, “Configuring Universal Agent to accept a FILE data provider” on page 337. Follow all the steps, with one change. When supplying the “FILE” data provider (as shown in Figure 8-32 on page 343), add “SCRIPT” to the end of line. The content of Universal Agent’s environment file is shown in Figure 8-61.



```
*------------------------------------------------------------------*
* Force specific non-standard tcpip domain name                    *
*------------------------------------------------------------------*
* KUM_TCPIP_DOMAIN_NAME=
*
*------------------------------------------------------------------*
* Messages (resource bundle) file directory                        *
*------------------------------------------------------------------*
KUMP_RB_PATH=C:\IBM\ITM\TMAITM6\rb
NLS1_LOCALEDIR=C:\IBM\ITM\TMAITM6\rb
*
*------------------------------------------------------------------*
* Needed for multi-home system (more than one network interface)   *
*------------------------------------------------------------------*
* KDCB0_HOSTNAME=
* KUM_DCH_HOSTNAME=
* KUM_DP_HOSTNAME=
* KUM_DCH_HOST=
* KUMP_DCH_HOST=
*
*------------------------------------------------------------------*
* UA Startup automatic start DP options                            *
* (ASFS,APIS,FILE,SOCK,HTTP,SNMP,POST,ODBC,SCRP)                   *
*------------------------------------------------------------------*
KUMA_STARTUP_DP=asfs,FILE,SCRIPT
KUMP_DP_FILE_SWITCH_CHECK_INTERVAL=30
```

Figure 8-61 Configuring the environment of Universal Agent for SCRIPT data provider

Universal Agent metafile for script data provider

In this section we describe the basic information that is necessary for configuring the Universal Agent to use a custom script.

Universal Agent can read the data from various sources. Based on data source, the metafile can have completely different syntax.

At least four rows are necessary for a metafile pointing to a script:

- Application name
- Attribute group name
- Source name (script path)
- At least one attribute name

As an example we provide the complete metafile that we use in this scenario. By importing the following metafile into Universal Agent, the Universal agent will:

- Periodically issue a script checking ITDWB Web application availability.
- Parse output of this script.

- Format the output into attributes.
- Send the attributes to the Tivoli Monitoring server.

Example 8-6 shows the metafile that we use in this scenario for periodical checks of ITDWB database availability.

Example 8-6 Metafile used for monitoring of ITDWB Web application

```
//APPL WEBSPHERE_CHECK_TDWB
//NAME WEBSPHERE_STATUS K 90 AddSourceName AddTimeStamp
//SOURCE SCRIPT websphere_check_tdwb.cmd
//ATTRIBUTES
status R 256 +FILTER={SCAN(0,TDWB is)}
```

Now we explain each row of the metafile:

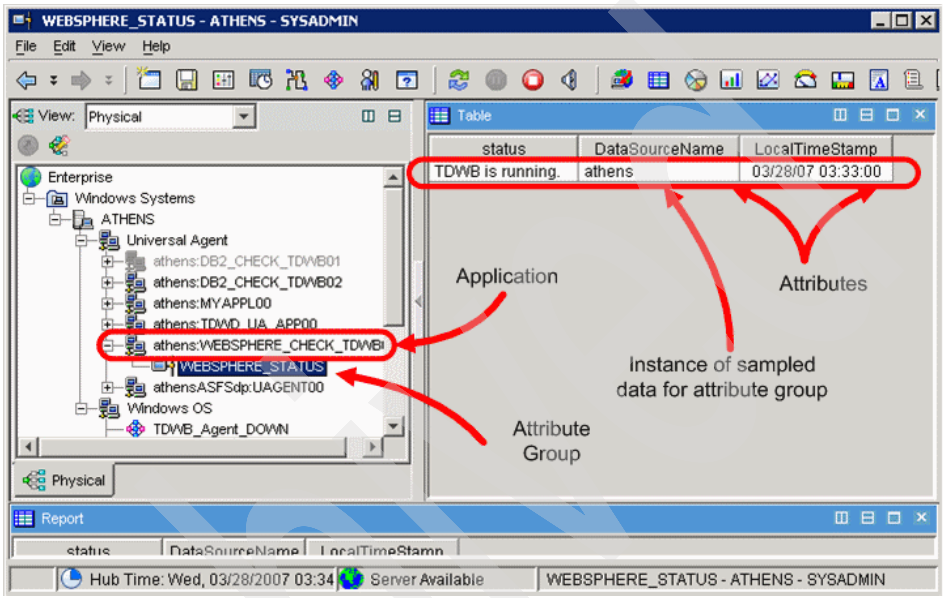
- //APPL WEBSPHERE_CHECK_TDWB

  Each metafile must contain an application with a unique name. Be aware that also the *first three characters* of the application name must be *unique* within Tivoli Monitoring environment. Another technical limitation is that the application name must not start with a K character because this character is reserved as the prefix for Tivoli Monitoring commands.

  The application name used in our example is WEBSPHERE_CHECK_TDWB. The name is correct because it does not start with K and because the first three characters (WEB) are a unique prefix within Tivoli Monitoring environment.

- //NAME WEBSPHERE_STATUS K 90 AddSourceName AddTimeStamp

  This row contains the name of an *attribute group*. An attribute group is nothing more than a set of attributes of various data types (strings, timestamps, integers, and so on). You can imagine a attribute group as a row in the table (for instance, on an event) with its columns (each value in a column is one attribute). An attribute group should contain at least one attribute, but often it is much more (timestamp, hostname, message, monitored entity status, and so on). An attribute group is represented in the Tivoli Enterprise Portal in a subtree of its application. Figure 8-62 shows how applications, attributes, and attribute groups are represented in the Tivoli Enterprise Portal.



status	DataSourceName	LocalTimeStamp
TDWB is running.	athens	03/28/07 03:33:00

Figure 8-62 Applications, attributes, and attribute groups within Tivoli Enterprise Portal

Additional parameters in this row have this meaning:

– K - keyed attribute group

– 90 - time to live (TTL). Specifies how long is each *attribute set* (one data sample) is alive. When this interval expires the attribute set vanishes from Tivoli Enterprise Portal. The value of TTL should be larger than the sampling interval, which is explained below.

– AddSourceName - adds an attribute that carries the host name of the monitored server.

– AddTimeStamp - adds the time when the sample was acquired.

► //SOURCE SCRIPT websphere_check_tdwb.cmd

In this row a source of sampled data is specified. For our purposes we use the SCRIPT keyword. If we wanted to monitor a text log file, we would use the FILE keyword.

Note: As you can see, the file name does not include the full path in this example. When the full path is not specified, the Universal Agent expects the file to be in the scripts subdirectory. On Windows the default path for Universal Agent scripts is C:\IBM\ITM\TMAITM6\scripts. If the script is located in another directory, you must supply the file name with the complete path.

- //ATTRIBUTES

  This is a delimiter announcing a beginning of an attributes definition. No additional parameters are necessary for this row.

- status R 256 +FILTER={SCAN(0,TDWB is)}

  We define an attribute with name *status*.

  Additional parameters in this row have the following meaning:

  – R - record. Take all the content of the scripts output.

  – 256 - maximal attribute size (string with maximal length of 256 characters).

  – +FILTER={SCAN(0,TDWB is)} - filtering the text output of the script. Get only those rows, which contain a string “TDWB is”. When searching for “TDWB is”, start at the beginning of each row. The plus(+) sign means that the filter is inclusive (it filters in the rows that match the pattern “TDWB is”).

    We search for string “TDWB is” because this pattern is contained in the only meaningful script, websphere_check_tdwb.cmd, which we have described before.

Save the metafile definition from Example 8-6 on page 381 into text file C:\IBM\ITM\TMAITM6\metafiles\websphere_check_tdwb.mdl.

Note: In fact, you can save the metafile into any directory you want, with a different file name and extension. Nothing is mandatory, but keeping the default paths and naming conventions makes any further maintenance easier.

We have completed the metafile. Now it is ready for validating and import into Universal Agent.

Validating and importing the metafile

Validation of a metafile is a process that checks whether the metafile does not contains syntax errors. If the metafile is correct, the validation may continue with importing the metafile into the Universal Agent definitions.

Note: All the commands must be issued from the directory of the Universal Agent. This directory corresponds to the CANDLE_HOME variable. On Windows it is C:\IBM\ITM. The path of Universal Agent is thus C:\IBM\ITM\TMAITM6. You *must* be in this directory to issue the Universal Agent commands.

Validation of metafile is performed with the following command:

`kumpcon validate` *`metafile_name`*

Example 8-7 shows the validation of the metafile websphere_check_tdwb.mdl. If the validation was successful, you are asked whether you want to import the application defined in this metafile into Universal Agent. To import the metafile, enter `i` and press Enter. After that the application is imported into the Universal Agent. When the application gets imported, it is active immediately.

Example 8-7 Validation and import of metafile definition

```
C:\IBM\ITM\TMAITM6>kumpcon validate websphere_check_tdwb.mdl
KUMPS001I Console input accepted.
KUMPV025I Processing input metafile
C:\IBM\ITM\TMAITM6\metafiles\websphere_check.mdl
KUMPV026I Processing record 0001 -> //APPL WEBSPHERE_CHECK_TDWB
KUMPV149I Note: APPL names starting with letters N-Z are designated for
customer UA solutions.
KUMPV026I Processing record 0002 -> //NAME WEBSPHERE_STATUS K 300
AddTimeStamp Interval=60
KUMPV026I Processing record 0003 -> //SOURCE SCRIPT
websphere_check_tdwb.cmd
KUMPV026I Processing record 0004 -> //ATTRIBUTES
KUMPV026I Processing record 0005 -> status R 256 +FILTER={SCAN(0,TDWB)}
KUMPV026I Processing record 0006 ->
KUMPV027I Blank input record skipped
KUMPV000I Validation completed successfully
KUMPV090I Application metafile validation report saved in file
C:\IBM\ITM\TMAITM6\metafiles\websphere_check_tdwb.rpt.

KUMPS065I Do you wish to Import or Refresh this metafile?
<Import/Refresh/Cancel>
i
```

```
KUMPS020I Import successfully completed for
websphere_check_tdwb.mdl
```

Note: If you specify the metafile name without the full path, it must be located in Universal Agent metafiles subdirectory. On Windows the default path for Universal Agent metafiles is C:\IBM\ITM\TMAITM6\metafiles. If the metafile is located in another directory, you must supply the file name with the complete path.

Now you have everything prepared on the agent side. Another steps must be done in the Tivoli Enterprise Portal.

Viewing the application in Tivoli Enterprise Portal

In previous steps we took these actions:

- Configured Universal Agent to be able to use the SCRIPT data provider.
- Created a monitoring script and stored it into the Universal Agents subdirectory dedicated for scripts.
- Created a metafile, pointing to that script and parsing its output. This metafile defines a new application and a new attribute group that will be visible in Tivoli Enterprise Portal. We stored the metafile in the Universal Agents subdirectory dedicated for metafiles.
- Validated the syntax of the metafile and imported it into the Universal Agent.

The result of these actions should be visible in the Tivoli Enterprise Portal. If you performed these actions while the Tivoli Enterprise Portal was running, you can see a message showing that the Navigator pane needs to be updated.

After you have refreshed the Navigator pane, you can see that a new application has been added. The name of the new application should consist of the Tivoli Dynamic Workload Broker server host name and the application name specified in the Universal Agent's metafile. The two-digit suffix displays the application version and modification number (initial value is 00). In our case the application name should be *<hostname>*:WEBSPHERE_CHECK_TDWB00.

Creating a view on monitored data

In this section we describe how to create a view that displays the monitored data.

A *view* displays the monitored data in the way we choose. It may be table, many sorts of graphs, or whatever Tivoli Enterprise Portal offers.

For our purposes a *table view* is sufficient. Defining that view requires a few steps:

1. In the Navigator pane click our application.
2. Click the table icon in the top of the Tivoli Enterprise Portal desktop. Your cursor changes and looks like the icon you just clicked. Move the cursor over the undefined workspace and click. Answer **Yes** if a question window `Assign the query now?` appears.
3. In the following window click **Click here to assign the query**.
4. The Query Editor window appears. In the left pane expand the **Universal Data Provider** branch. After that expand our application branch. After that expand the attribute group branch and navigate to the deepest level of this branch. The right pane will show all the attributes defined within the attribute group of our application. Select or deselect the attributes so that you customize what attributes you want to have your view.
5. Click **OK** in the Query Editor. Also confirm the following window.

You have created a view displaying the monitored data.

Setting up thresholds

In this section we describe how to set up thresholds distinguishing displayed data by severity.

You can see the monitored data, but they are not distinguished by severity. Tivoli Monitoring server does not know yet which data carry information about something harmless, and which data report serious errors.

To distinguish the way in which the data are represented, you must define *thresholds* for a particular view. Thresholds determine which data should be assigned specified severity (informational, warning, or critical).

To define thresholds:

1. Right-click anywhere in the view and select **Properties**.
2. Click **Thresholds**. Each row represents the threshold for a specified condition. Each row has more columns, and there is an AND relationship among them. This means that the of threshold condition is fulfilled only if *all* the conditions in a row are true.

   The first column determines the severity. Possible choices are informational, warning, and critical. Any of the next columns represents one condition. There must be at least one condition set for the threshold to become active.

   In our scenario we want to set the critical threshold on the status `TDWB is NOT running!`

In the first column leave severity set to critical. In the second column select **Scan for string within a string** function and the **==** (equals) operand. Then fill in the word `NOT` into the cell.

The word NOT is sufficient for the formula. The monitoring script returns two possible values of attribute *status*:

- TDWB is running.
- TDWB is *not* running.

Because we compare only a substring from output (we use the option "Scan for string within a string"), it is enough if we specify only the unique part of the string.

3. Click **OK** and refresh the view. You should see that the corresponding rows have changed.

Important: This section describes how a particular view was customized to represent monitored data. This customization has nothing to do with Tivoli Monitoring *situations* at all. Customizing a view gives you only a possibility off how to distinguish data for operators. But unless you define a situation (even with the same conditions that you used for threshold setting in a view), you cannot see the *bad things* from the global perspective. You must have the particular view put in your current workspace if you want to be aware of its events. Furthermore, setting a threshold in a view does not allow you to take an automatic corrective action.

Creating a situation

In this section we describe the steps that must be taken in order to configure Tivoli Monitoring to fire a situation when a a custom script returns output `TDWB is NOT running!!`

Note: In this section we list the steps necessary for creating the situation without snapshots. The detailed step-by-step instrucions for creating a situation, including snapshots, are in "Monitoring of DB2 availability on Windows" on page 359.

1. In the Navigator pane, go to the Universal Agent under the monitored machine (Tivoli Dynamic Workload Broker server). Expand the **Universal Agent** branch and select the application name that we defined in previous steps. In our case the name is *<hostname>*:WEBSPHERE_CHECK_TDWB00. Right-click the application and select **Situations**.

2. The situation editor appears. Click **Create new situation** in the upper left corner. In the window that pops up fill in the situation name `TDWB_Server_DOWN`, and from the drop-down list select **Universal Agent**. Click **OK**.
3. We want to create situation that will be fired when the monitoring script returns output `TDWB is NOT running!!` First we need to select attributes that will be used for evaluating the situation formula (when to fire a situation). In the left pane select the source attribute group. In our case it is WEBSHERE STATUS. In the right pane select attribute **Status**.
4. In the situation formula editor set the following evaluation criteria: status: **Scan for string within a string ==** NOT.

   The word NOT is sufficient for the formula. The monitoring script returns two possible values of attribute *status*:

   – TDWB is running.
   – TDWB is NOT running!!

   Because we compare only a substring from output (we use the option "Scan for string within a string"), it is enough if we specify only the unique part of the string.

   Figure 8-63 shows the situation formula.
5. Optionally, adjust the sampling interval. This interval specifies how often the Windows OS agent will check the service status and send the results to the Tivoli Monitoring server.

   The value of the sampling interval can be any that fits your needs. We have used 30 seconds in our scenario, but the recommended sampling interval is usually a bit longer (from 2 to 5 minutes). Too short of a sampling interval produces unnecessary load, and too long of an interval can discover a possible failure too late.

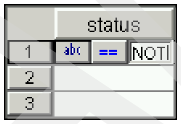


Figure 8-63 Situation formula for monitoring of ITDWB enterprise application availability

Now you have done all the necessary steps for creating a situation that will fire when the monitoring script returns result `TDWB is not running!!`

For more detailed step-by-step instructions (including snapshots), see a similar scenario describing monitoring of the DB2 availability in "Monitoring of DB2 availability on Windows" on page 359.

Setting up corrective action

Now we describe how to set up a corrective action to the situation that we have defined in the previous section.

In this case we want to start the ITDWB enterprise application installed in the WebSphere Application Server. We use the same mechanism for starting the enterprise application as for checking its state. We use a **`wsadmin`** command with its instruction file for making calls directly into the WebSphere Application Server. For more details about the **`wsadmin`** command see "Custom monitoring scripts for WebSphere Application Server" on page 377 and "Calling the instructions for wsadmin script from command line" on page 379.

First we need to create an instruction file for the **`wsadmin`** administration command. We chose the jacl programming language. The content of the file is displayed in Example 8-8.

Important: In Example 8-8 on page 389 we use hard-coded values for *cell*, *node*, and *server* attributes. You must modify the values to correspond to your WebSphere Application Server installation. At the least you have to change the host name included in the values for *node* and *cell*. Consult your WebSphere Application Server administrator if unsure about the correct values.

Example 8-8 Jacl script for starting the enterprise application

```
set appManager [$AdminControl queryNames \
cell=athensNode01Cell,node=athensNode01,type=ApplicationManager,process
=server1,*]
$AdminControl invoke $appManager startApplication ITDWB
```

Note: The first line ends with a backslash (\). This is a typical UNIX convention instructing the interpreter that the line is not completed and continues on the following line.

Save the content into file C:\IBM\ITM\TMAITM6\scripts\websphere_start_tdwb.jacl.

Example 8-9 contains the corresponding Windows script that calls the **`wsadmin`** command with its instruction file.

Example 8-9 Windows script used to call wsadmin interface

```
@echo off
SET SystemRoot=C:\WINDOWS
c:\progra~1\ibm\websphere\appserver\bin\wsadmin -f c:\ibm\itm\tmaitm6\s
cripts\websphere_start_tdwb.jacl
```

Save the content into file C:\IBM\ITM\TMAITM6\scripts\websphere_start_tdwb.cmd.

Now you have prepared the necessary scripts that will be invoked by the Tivoli Monitoring automatic corrective action.

Switch to the Tivoli Enterprise Portal and do the following:

1. In the Navigator pane, navigate to the Universal Agent under the monitored machine (Tivoli Dynamic Workload Broker server). Expand the **Universal Agent** branch and select the application name for which we have defined the situation. The application name looks like this *<hostname>*:WEBSPHERE_CHECK_TDWB00.
2. Right-click the application and select **Situations**.
3. In the next window navigate in the left pane to the **TDWB_Server_DOWN** situation. Select it and in the right pane select **Take action**.
4. Type `C:\IBM\ITM\TMAITM6\scripts\websphere_start_tdwb.cmd` into the textbooks System Command.
5. Make sure that you have **Execute Action at the Managed system (Agent)** selected. Confirm by clicking **OK**.

Testing the situation and corrective action

To test whether the ITDWB enterprise application gets restarted after it has crashed (or was stopped), do the following steps.

Note: We strongly recommend doing the following steps *only* in a *testing* environment. Do not stop any service in a production environment unless you have a maintenance window arranged.

1. Log on to the WebSphere Application Server Administrative Console (open the Web browser and go to http://*<tdwb_server_hostname>*:9060/ibm/console).

2. In the left pane expand **Applications** and then select **Enterprise Applications**. Select the **ITDWB** checkbox. Click **STOP** in the menu above. The status of the ITDWB application will change from running to stopped.
3. Wait for the same interval that you specified when defining the situation. After that click again **Enterprise Applications** in the left pane. The state of ITDWB enterprise application should change back to *running*.

8.4.13 Default values and file locations

This chapter lists default values, such as the locations of important files, values passed to Tivoli Dynamic Workload Broker - Tivoli Monitoring integration script, and so on.

Default values on Windows platforms

Default paths and other values for Windows platforms are listed in Table 8-3.

Table 8-3 Default monitoring related values on Windows

Name	Default value
Script for sourcing an environment	C:\Program Files\IBM\ITDWB\bin\tdwb_env.bat
Integration script	C:\Program Files\IBM\ITDWB\bin\tepconfig.bat
Tivoli Dynamic Workload Broker TEPListener configuration file	C:\Program Files\IBM\ITDWB\config\TEPlistener.properties
Metafile definition for Universal Agent so it is able to parse the messages in the TDWB log file	C:\Program Files\IBM\ITDWB\Server\TEP\TDWBMeta_Sample.mdl
ITM default directory (corresponding to CANDLE_HOME)	C:\IBM\ITM
Universal Agent home directory	C:\IBM\ITM\TMAITM6
Universal Agent binary	C:\IBM\ITM\TMAITM61\kuma610.exe
Universal Agent console command (import, validation, and so on)	C:\IBM\ITM\TMAITM61\kumpcon.exe
Universal Agent metafile path	C:\IBM\ITM\TMAITM6\metafiles
Universal Agent script path	C:\IBM\ITM\TMAITM6\scripts

Default values on UNIX platforms

Default paths and other values for UNIX platforms are listed in Table 8-4.

Table 8-4 Default monitoring related values on UNIX

Name	Default value
Script for sourcing of environment variables	. /opt/IBM/ITDWB/Server/bin/ tdwb_env.sh
Integration script	/opt/IBM/ITDWB/Server/bin/ tepconfig.sh
Tivoli Dynamic Workload Broker TEPlistener configuration file	/opt/IBM/ITDWB/Server/config/TEPlistener.properties
Metafile definition for Universal Agent, so it is able to parse the messages in the TDWB log file	/opt/IBM/ITDWB/Server/TEP/ TDWBMeta_Sample.mdl
ITM default directory (corresponding to CANDLE_HOME)	/opt/IBM/ITM
Universal Agent home directory	/opt/IBM/ITM/TMAITM6
Universal Agent binary	/opt/IBM/ITM/TMAITM61/kuma610
Universal Agent console command (import, validation, and so on)	/opt/IBM/ITM/TMAITM61/kumpcon
Universal Agent metafile path	/opt/IBM/ITM/TMAITM6/metafiles
Universal Agent script path	/opt/IBM/ITM/TMAITM6/scripts

List of all possible states of Tivoli Dynamic Workload Broker jobs

- FAILED
- SUCC
- RES_ALLOC_RECEIVED
- EXEC
- RES_ALLOC_FAILED
- UNKNOWN
- NOT_EXECUTED
- WAIT_FOR_RES
- SUBMITTED_TO_ENDPOINT
- SUBMITTED
- PENDING_CANCEL
- CANCEL_ALLOC
- RES_REALLOC

- RES_REALLOC_FAILED
- CANCEL

Default list of TDWB job states written into log file

By default, only the following states are logged into the log file:

- FAILED
- RES_ALLOC_FAILED
- NOT_EXECUTED
- RES_REALLOC_FAILED
- CANCEL

For troubleshooting Tivoli Monitoring integration refer to 10.3, “Troubleshooting the integration with IBM Tivoli Monitoring” on page 504.

9

Interacting with Tivoli Dynamic Workload Broker using the Web services interface

The intent of this chapter is to provide a description that allows you to integrate your business applications with Tivoli Dynamic Workload Broker. Business applications running in your environment may require you to submit external jobs.

The following are discussed in this chapter:

9.1 Why you would use the Web services interface

Selecting the right candidate for the external job in heterogeneous and distributed environment is not an easy task. Usually it requires additional efforts of developers to implement logic evaluating where to launch a specific external job based on its requirements. Integration with the Tivoli Dynamic Workload Broker could solve this task quickly and efficiently.

Tivoli Dynamic Workload Broker is a complex application that relies on the service-oriented architecture (SOA). It provides the Web services interface that allows your business applications to programmatically leverage the job management capabilities provided by Tivoli Dynamic Workload Broker. You can easily incorporate Tivoli Dynamic Workload Broker into the *service-oriented architecture (SOA)* in your environment.

You can easily extend your business applications by job brokering capabilities by using the Web services interface provided with Tivoli Dynamic Workload Broker. Your business applications may focus on their own logic. Management of external jobs (that are dependent on suitable resources) is passed to the Tivoli Dynamic Workload Broker. It will find the best suitable resource for running the job and will manage the job's life cycle. The application integrated with the Tivoli Dynamic Workload Broker through the Web services interface can perform similar operations, as the operator using the Tivoli Dynamic Workload Broker command-line interface.

In this chapter we describe the mechanism of how to access the Tivoli Dynamic Workload Broker server from external applications using the Web services technology.

We briefly describe the concept of Web services, list the available Tivoli Dynamic Workload Broker Web services, and describe their operations. Finally, we provide a scenario describing how to build a sample client that integrates with Tivoli Dynamic Workload Broker through the Web services interface.

It is beyond the scope of this document to provide a detailed description of the Web services technology and related programming techniques. For a detailed description of the Web services architecture visit http://www.w3.org/TR/ws-arch/ or read *Web Services Handbook for WebSphere Application Server 6.1,* SG24-7257.

Important: In this chapter we describe the Web services interface available with Tivoli Dynamic Workload Broker V1.2. This release (planned general availability in June 2007) runs on WebSphere Application Server V6.1, since the Web services interface does not work with Tivoli Dynamic Workload Broker V1.1.

For building an application leveraging Tivoli Dynamic Workload Broker Web services interface, you should use at least Tivoli Dynamic Workload Broker V1.2 installed on WebSphere Application Server V6.1. Otherwise, your application may not function correctly. We performed our tests for this chapter on a beta version of Tivoli Dynamic Workload Broker V1.2.

9.2 Web services concepts

In this chapter we focus on those terms and technologies that are necessary for building a client leveraging the Web services interface provided by the Tivoli Dynamic Workload Broker server.

We do not describe the Web services in detail or list all of the underlying technologies. We just provide an overview of the techniques that we use in our scenario.

9.2.1 Brief description of Web services

While there is no commonly accepted definition for Web services, we can try to define a Web services concept from two different perspectives.

From the business perspective Web services are a mechanism that brings *service-oriented architecture* into life. Business applications can easily integrate with another business applications and thus maximize their potential. By using the Web services concepts, the applications can be organized more from the business perspective rather than from the IT perspective. Coupling the applications through Web services allows rapid integration of heterogeneous components and thus significantly reduces the investments necessary for integration development.

From the technical perspective Web services are one of the implementations of the Distributed Objects architecture. In addition to the Distributed Objects architecture, Web services offer several features that make them the best candidate for implementing the service-oriented architecture.

In the following sections we briefly explain the basic concepts of distributed objects and then we mention the advantages that make Web services more competitive and efficient for today's world's needs.

Distributed objects architecture

From the technical point of view Web services are just another concept implementing the distributed objects architecture, such as Common Object Request Broker Architecture (CORBA), Common Object Model (COM), or Remote Method Invocation (RMI).

The architecture of distributed objects allows us to call remote objects (objects that are running in a different address space) by a local objects *as if they were running locally* (in the same address space as the local objects). This is implemented by propagating interfaces of the remote object as *skeleton* and *stub*. Skeleton is an interface on the remote side (typically server side) and stub is an interface on the local side. Stub and skeleton communicate through the network using a defined protocol and messaging standard.

Now we describe the data flow that occurs when a local object requests a response from the remote object. The request can be as simple as "compute 2+2 and give me a result" or a query, which may result in complex response. It depends on the application logic.

Distributed objects - data flow

The data flow in Distributed Objects architecture is:

1. The local application (typically client) has stub code incorporated and calls the methods (commands, actions) that are defined by the stub. The client *thinks* that he communicates with the real object implementation.
2. The stub translates the request into the *message* and transfers the message thought the network to the skeleton (remote interface of the remote object).
3. The skeleton then invokes the desired method on the real object instance.
4. Within the remote object instance the internal logic is computed (for instance, "return result of a+b").
5. The result is passed back to the skeleton. The skeleton translates the result into the message and sends it through the network to the stub.
6. The stub then reads the message and converts it to the response for the requestor (client application object).
7. The requestor processes the response. All of the above described communication is hidden from the local object (requestor). The local object was *thinking* that he directly called the instance of the remote object.

Web services - implementation of Distributed Objects

Now when we describe the distributed objects, we repeat that Web services is just one of the possible implementations of this Distributed Objects architecture. Unlike the CORBA and COM, Web services has more capabilities. The most important of them are:

- Web services has unified messaging syntax based on XML.
- Web services is in the majority leverage HTTP (HTTPS) protocol, so it is very easy to make it work through firewalls.

  From the technical perspective this topic is very important. Issues with ports opened during the communication among remote components are important disadvantages of other Distributed Objects implementations.
- Web services is indenpendent on platform and on programming language used for applications it is integrating. Principally, the application written in C++ running on a Linux server can communicate with another application written in Java and running on a Windows system.
- There are commonly accepted techniques that implement a registry of Web services. A term related to this topic is *Service Broker*. However, no final unified standard exists at the time of publishing this book. Currently, a Web Services Inspection Language (WSIL) and Universal Description, Discovery and Integration (UDDI) implement the service registry.
- Web services is widely supported by the major vendors.

Web services leverages many industry standards, such as XML, JAX-RPC, SOAP, and so on. For the purposes of this chapter (building a sample client communicating with Tivoli Dynamic Workload Broker server through Web services) only a basic understanding of the following technologies is sufficient:

- Extensible Markup Language (XML) - the generic markup language that can be used to describe any kind of content in a structured way. Unlike HTML, the XML does not focus on the presentation layer, it just describes the data in the structured way by using nested elements.
- Simple Object Access Protocol (SOAP) - is a lightweight platform indenpendent *messaging protocol*. The syntax of SOAP is based on XML. Web services leverages SOAP as its messaging protocol.
- Java API for XML-based Remote Procedure Call (JAX-RPC) - is an application interface that allows calling remote procedures using XML (SOAP) messages. Leveraging JAX-RPC allows the programmer to call just the JAX-RPC API, and it performs all of the necessary tasks for transforming the requests/responses into and from SOAP messages.
- Web Services Description Language (WSDL) - is a language for defining services as a collection of endpoints that are capable of exchanging

messages. WSDL is also an XML-based language. Operations provided by the Web services, their parameters, and data types are described in WSDL. We provide many examples in this chapter that are written in WSDL.

There is one more term that you should be familiar with before you start to develop the client for the Tivoli Dynamic Workload Broker server. This term is *JSDL*, which is an acronym of *Job Submission Definition Language*. JSDL is an XML-based language that is used for defining jobs for the Tivoli Dynamic Workload Broker. We use one JSDL file in our scenario. The Job Submission Definition language is described in Chapter 4, “Working with Tivoli Dynamic Workload Broker” on page 141. A detailed JSDL reference is included in *IBM Tivoli Dynamic Workload Broker User's Guide Version 1.1*, SC32-2281.

9.3 Deeper view of jobs in Tivoli Dynamic Workload Broker

In this section we describe the possible states of jobs that were submitted to the Tivoli Dynamic Workload Broker server. We provide a state diagram showing the job life cycle and explain the job states. We also briefly describe the job management actions that can be called from the client.

This knowledge is useful for developing the integration interfaces of your business applications with the Tivoli Dynamic Workload Broker through the Web services interface.

9.3.1 Job definitions

All Tivoli Dynamic Workload Broker jobs are defined in the Job Submission Definition Language. In the following sections we reference this language by its acronym, JSDL.

JSDL is an XML-based language. The complete schema reference for the JSDL can be found in *IBM Tivoli Dynamic Workload Broker User's Guide Version 1.1*, SC32-2281.

There are two different approaches that can be used for storing the job definitions:

- Storing the job definitions on the file system into files, typically with the .jsdl extension.
- Storing the job definitions in the Job Repository on the Tivoli Dynamic Workload Broker server. The Job repository is physically represented by a couple of tables in database TDWB defined in the RDBMS.

Note: The Tivoli Dynamic Workload Broker V1.2 supports IBM DB2, as well as Oracle RDBMS. The Tivoli Dynamic Workload Broker V1.1 only supports DB2.

9.3.2 Job life cycle within Tivoli Dynamic Workload Broker

In this section we show the job life cycle within Tivoli Dynamic Workload Broker. We provide a state diagram showing the possible job states and the defined changes among the job states.

Figure 9-1 shows the flow of job states. The job states marked with an asterisk (*) are final states.

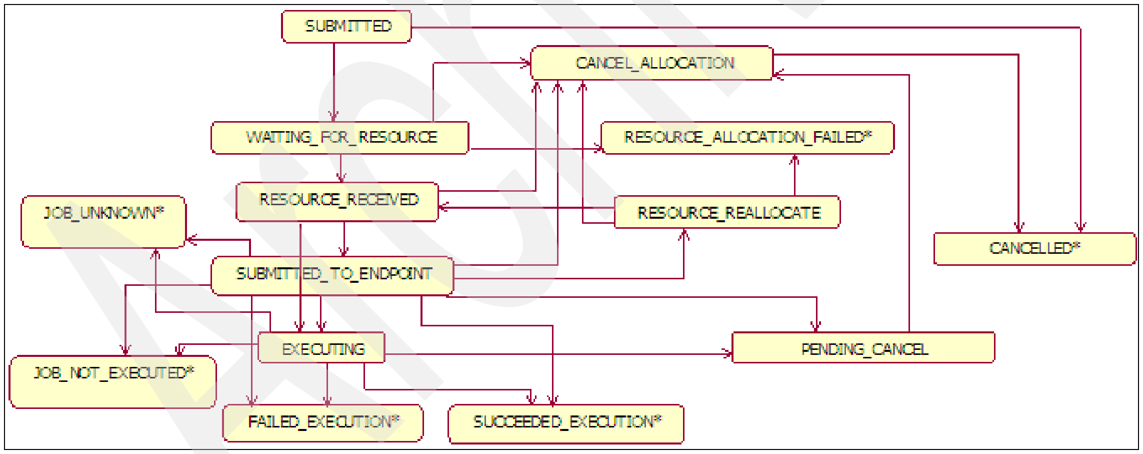


Figure 9-1 Job life cycle within Tivoli Dynamic Workload Broker server

Table 9-1 provides a list of possible job states and their descriptions.

Table 9-1 Possible job states

Job state	Description
Submitted	The Job Dispatcher has accepted and persisted a new job request for execution from the submitter.
Waiting_for_Resource	A request for endpoint execution has been submitted to the Resource Advisor component and the job is waiting for notification and assignment of execution resources.
Canceled	The submitting user may canceled the job at any stage of execution life cycle.
Resource_Received	The Resource Advisor returned the target execution resources.
Submitted_to_Endpoint	The requested job is submitted to the target execution endpoint.
Resource_Reallocate	The Job Dispatcher decides to return the endpoint to the Resource Advisor and request a new endpoint because the submission to the current endpoint failed due to a connection error.
Executing	The Job Executor on the target endpoint started job execution.
Pending_cancel	The Job Dispatcher sent the cancel request to the job executor and waits for the feedback of the actual cancellation.
Failed_Execution	The executing job returned a failure (return code not equal to 0).
Succeeded_Execution	The job successfully completed execution at the endpoint.
Unable_to_start	The job cannot be started due to authentication, path problems.
Cancel_allocation	The Job Dispatcher received the cancellation feedback from the endpoint and requested the resource allocation cancellation to the Resource Advisor.
Job_unknown	The job state returned by the agent to the server if the agent was not able to find the status of the actual job.
Allocation_Failed	The Resource advisor cannot find any available resource matching the job requirements.

We use the job status as the evaluation criteria in our sample scenario.

9.3.3 Client interactions

In this section we briefly describe how the client application is able to interact with the Tivoli Dynamic Workload Broker through the Web services interface.

Figure 9-2 displays the possible operations provided by the Tivoli Dynamic Workload Broker through the Web services interface.

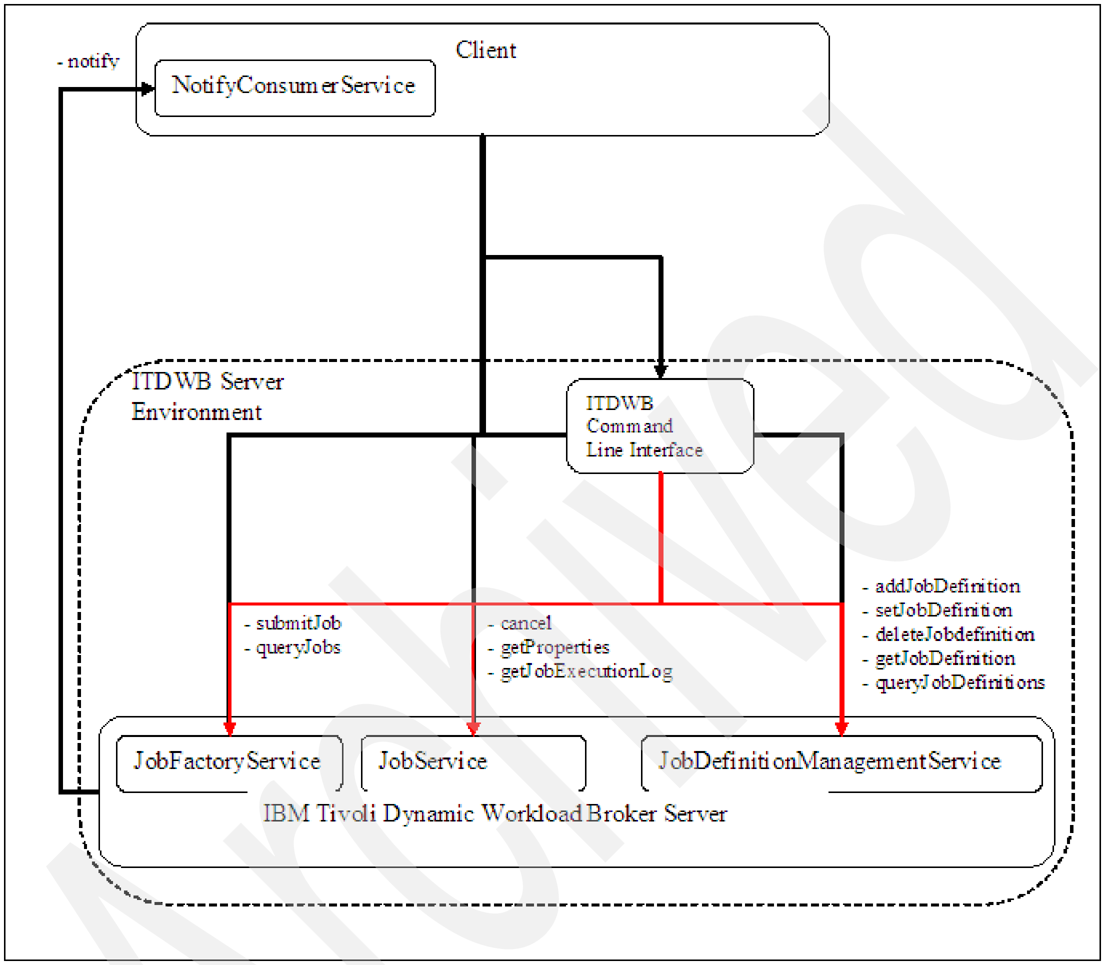


Figure 9-2 Possible client interactions with Tivoli Dynamic Workload Broker jobs

9.4 Web services interfaces provided by the Tivoli Dynamic Workload Broker server

In this section we describe the services and operations provided by the Tivoli Dynamic Workload Broker server.

We focus on the following actions that can be done by invoking the Tivoli Dynamic Workload Broker Web services:

- Job Factory Service
  - Submitting jobs
  - Querying job
- Job Service
  - Cancelling jobs
  - Getting job properties
  - Getting job output
- Job Definition Management Service
  - Adding job definitions
  - Modifying job definitions
  - Deleting job definitions
  - Querying job definitions

9.4.1 How to read this section

The following content describes the Web services interface provided with the Tivoli Dynamic Workload Broker. We create a special *section* for each *Web Service* (Job Factory, Job, and Job Definition Management Service). Each of these sections is divided into subsections, which contain the description of operations of a particular Web Service and also a response to that operation, if available.

In each section we first provide:

- Short description of the Web Service itself
- Extract from the WSDL file describing the Web Service with its operations

In each corresponding subsection we provide:

- Extract from the WSDL file describing the type definitions for the particular operations (if available; some operations do not take arguments)
- Extract from the WSDL file describing the type definitions for response to the invocation of the specific operation (if available; some operations do not have a response)

  The type definitions of particular operations are in fact the parameters (arguments) that you will pass to the methods, when writing your program. The type definitions describe what method uses which parameters, when invoked in your program.

  The type definitions of responses to particular operations are the parameters that come back with the response.

Note: Some type definitions may include elements for *variables* and *affinity*. The description of variables and job affinity is included in 9.4.5, “Important terms related to job definitions” on page 430.

All the WSDL extracts originate from the corresponding WSDL files. They are:

- Job Factory Service - JobFactory.wsdl
- Job Service - Job.wsdl
- Job Management Definition Service - JobDefinitionMgmt.wsdl

9.4.2 Job Factory service

In this section we describe the JobFactory service. This service provides following functionality:

- Submitting jobs
- Querying job status

Example 9-1 contains the extract from the JobFactory.wsdl file. You can see all of the operations defined with this Web Service.

Example 9-1 JobFactory Web Service and its operations for submitting and querying jobs

```
<wsdl:portType name="JobFactory">

       <wsdl:operation name="submitJobFromJSDL">
                <wsdl:input name="SubmitJobFromJSDLRequest"
message="jmjf:SubmitJobFromJSDLRequestMessage" />
                <wsdl:output name="SubmitJobFromJSDLResponse"
message="jmjf:SubmitJobFromJSDLResponseMessage" />
                <wsdl:fault name="OperationFailedFault"
message="jmjf:OperationFailedFaultMessage" />
                <wsdl:fault name="InvalidArgumentsFault"
message="jmjf:InvalidArgumentsFaultMessage" />
                <wsdl:fault name="ServiceUnavailableFault"
message="jmjf:ServiceUnavailableFaultMessage"/>
          </wsdl:operation>

      <wsdl:operation name="submitJobFromName">
                <wsdl:input name="SubmitJobFromNameRequest"
message="jmjf:SubmitJobFromNameRequestMessage" />
                <wsdl:output name="SubmitJobFromNameResponse"
message="jmjf:SubmitJobFromNameResponseMessage" />
                <wsdl:fault name="OperationFailedFault"
message="jmjf:OperationFailedFaultMessage" />
                <wsdl:fault name="InvalidArgumentsFault"
message="jmjf:InvalidArgumentsFaultMessage" />
```

```
                <wsdl:fault name="ServiceUnavailableFault"
message="jmjf:ServiceUnavailableFaultMessage"/>
      </wsdl:operation>

      <wsdl:operation name="queryJobs">
                <wsdl:input name="QueryJobsRequest" message="jmjf:QueryJobsRequestMessage" />
                <wsdl:output name="QueryJobsResponse" message="jmjf:QueryJobsResponseMessage"
/>
                <wsdl:fault name="OperationFailedFault"
message="jmjf:OperationFailedFaultMessage" />
                <wsdl:fault name="InvalidArgumentsFault"
message="jmjf:InvalidArgumentsFaultMessage" />
                <wsdl:fault name="ServiceUnavailableFault"
message="jmjf:ServiceUnavailableFaultMessage"/>
         </wsdl:operation>

      </wsdl:portType>

      <wsdl:service name="JobFactoryService">
         <wsdl:port binding="jmjf:JobFactoryBinding" name="JobFactory">
            <wsdlsoap:address location="http://localhost:9080/JobWS/services/JobFactory" />
         </wsdl:port>
      </wsdl:service>
```

Submitting jobs

In this subsection we describe the operations of the Job Factory Web Service that can be used for job submission.

There are two possible ways to submit a job:

- Submit a job from the definition stored in the Job Repository.
- Submit a job from the definition originating on the client side (either read from the JSDL file or created by the client application).

The submit operations return a Job EndPointReference (EPR) containing a Job Handle Unique ID. The Job EPR is used for all subsequent Job Service operations (like cancelling the job, getting job properties, and getting job output).

Submitting a job to the server can be either successful or unsuccessful. The reasons why the job submission may not succeed are:

- Submission fails to pass security checks.
- JSDL validation against schema was not successful (JSDL document is not valid).
- Incorrect variable substitution was performed.

In all of the cases listed above, the attempt to submit a job will return a *fault*.

Job executions are processed asynchronously. Return is made to the service client before the asynchronous processing is started. Job execution status will be sent to the client through the JobNotify service (as shown in Figure 9-1 on page 401). The description of the JobNotify service and corresponding details is included in 9.4.6, “Getting notified about job state changes” on page 431.

Now we describe the operations defined with the JobFactory service.

SubmitJobFromJSDL

Example 9-2 contains the type definitions for the submitJobFromJSDL operation. The client application must have a reference to the object with the JSDL definition available when invoking this Web service. The object containing the definition is constructed by application logic — either by reading from the JSDL file, or by creating the definition on-the-fly. In order to create the object containing the job definition it is necessary to create the JSDL document root structure first and then add all of the nested elements to that structure.

Example 9-2 Type definitions for submitJobFrom JSDL operation

```
<xsd:element name="submitJobFromJSDL">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="JobDefinitionDocument" minOccurs="1" maxOccurs="1">
                        <xsd:complexType>
                              <xsd:sequence>
                                    <xsd:element ref="jsdl:jobDefinition" minOccurs="1"
                                    maxOccurs="1"/>
                              </xsd:sequence>
                        </xsd:complexType>
                  </xsd:element>
                  <xsd:element name="Alias" type="xsd:string" minOccurs="0" maxOccurs="1" />
                  <xsd:element name="ClientNotifyEPR" type="wsa:EndpointReferenceType"
                  minOccurs="0" maxOccurs="1" />
                  <xsd:element name="Variable" type="jmjf:VariableType" minOccurs="0"
                  maxOccurs="unbounded"/>
                  <xsd:element name="Affinity" type="jmjf:AffinityType" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="SubmitterType" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
            </xsd:sequence>
      <xsd:anyAttribute />
</xsd:complexType>
</xsd:element>
```

The SubmitJobFromJSDL consists of:

- Job definition document - contains the JSDL of the job to be executed.
- Alias - alias of the job to which the submitting job should be affine.
- Client Notification EndpointReferenceType - address of the client implementing the *Notify* Web service described later. Messages about job state changes are delivered to the client that wants to be notified. Such a client (implementing the Notify Web Service) should be up and running at this address. In case the client is not available, the Tivoli Dynamic Workload Broker server will retry sending the notification of the latest status change until an interval specified in the Job Dispatcher expires.
- Variable - the run-time variables.
- Affinity - the job affinity specifications.
- SubmitterType - any string identifying the type of submitter (for instance WebUI, CLI, API, MyApp, and so on).

SubmitJobFromJSDLResponse

Example 9-3 contains the type definitions for the response to the submitJobFromJSDL operation.

Example 9-3 Type definitions for response to submitJobFromFile operation

```
<xsd:element name="submitJobFromJSDLResponse">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:elementname="JobEPR"type="wsa:EndpointReferenceType"minOccurs="1"
                  maxOccurs="1" />
                  </xsd:sequence>
            <xsd:anyAttribute />
      </xsd:complexType>
</xsd:element>
```

The SubmitJobFromJSDLResponse consists of the job EndpointReferenceType, the EndpointReferenceType of the job just created through the submission. Any subsequent operation like cancel, get job properties, and get job output should be made on this address because this address contains the job ID in the reference properties.

SubmitJobFromName

Example 9-4 contains the type definitions for the submitJobFromName operation. The job definition of the desired job must be stored in the job repository prior to invoking this operation.

Example 9-4 Type definitions for submitJobFromName operation

```
<xsd:element name="submitJobFromName">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="JobDefinitionName" type="xsd:QName" minOccurs="1"
                  maxOccurs="1" />
                  <xsd:element name="Alias" type="xsd:string"minOccurs="0"maxOccurs="1"/>
                  <xsd:element name="ClientNotifyEPR" nillable="true"
                  type="wsa:EndpointReferenceType"minOccurs="0" maxOccurs="1"  />
                  <xsd:element name="Variable" type="jmjf:VariableType" minOccurs="0"
                  maxOccurs="unbounded" />
                  <xsd:element name="Affinity" type="jmjf:AffinityType" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="SubmitterType" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
            </xsd:sequence>
            <xsd:anyAttribute />
      </xsd:complexType>
</xsd:element>
```

The SubmitJobFromName consists of:

- Job Definition Name - contains the name of the job definition to be executed. The job definition must already be stored in the job repository.
- Alias - alias of the job to which the submitting job should be affine.
- Client Notification EndpointReferenceType - address of the client implementing the *Notify* Web service described later on. Messages about job state changes are delivered to the client that wants to be notified. Such a client (implementing the Notify Web service) should be up and running at this address. In case the client is not available, the Tivoli Dynamic Workload Broker server retrys sending the notification of the latest status change until an interval specified in the Job Dispatcher expires.
- Variable - the run-time variables.
- Affinity - the job affinity specifications.
- SubmitterType - any string identifying the type of submitter (for instance, WebUI, CLI, API, MyApp, and so on).

SubmitJobFromNameResponse

Example 9-5 contains the type definitions for the response to the submitFromName operation.

Example 9-5 Type definitions for response to submitJobFromName operation

```
         <xsd:element name="submitJobFromNameResponse">
               <xsd:complexType>
                     <xsd:sequence>
                           <xsd:element name="JobEPR" nillable="false"
type="wsa:EndpointReferenceType" minOccurs="1" maxOccurs="1" />
                     </xsd:sequence>
                     <xsd:anyAttribute />
               </xsd:complexType>
         </xsd:element>
```

The SubmitJobFromNameResponse consists of the job EndpointReferenceType, the EndpointReferenceType of the job just created through the submission. Any subsequent operation like cancel, get job properties, and get job output should be made on this address that contains the job ID in the reference properties.

SubmitJobFromJSDLXml

Example 9-6 contains the type definitions for the submitJobFromName operation. The job definition of the desired job must be available to the application as a *string* object prior to invoking this operation. The string object can be constructed in any way, for instance, by reading the content of a JSDL file into the StringBuffer instance and then retyping it to string.

Example 9-6 Type definitions for submitJobFromJSDLXml operation

```
<xsd:element name="submitJobFromJSDLXml">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="JobDefinitionDocument" type="xsd:string" minOccurs="1"
maxOccurs="1" />
         <xsd:element name="Alias" type="xsd:string" minOccurs="0" maxOccurs="1" />
         <xsd:element name="ClientNotifyEPR" type="wsa:EndpointReferenceType" minOccurs="0"
maxOccurs="1" />
         <xsd:element name="Variable" type="jmjf:VariableType" minOccurs="0"
maxOccurs="unbounded" />
         <xsd:element name="Affinity" type="jmjf:AffinityType" minOccurs="0" maxOccurs="1" />
         <xsd:element name="SubmitterType" type="xsd:string" minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

The SubmitJobFromJSDLXml consists of:

- Job Definition Document - the reference to the string object containing the JSDL definition.
- Alias - alias of the job to which the submitting job should be affine.
- Client Notification EndpointReferenceType - address of the client implementing the *Notify* Web service described later. Messages about the job state changes are delivered to the client that wants to be notified. Such a client (implementing the Notify Web service) should be up and running at this address. In case the client is not available, the Tivoli Dynamic Workload Broker server retrys sending the notification oflatest status change, until an interval specified in the Job Dispatcher expires.
- Variable - the run-time variables.
- Affinity - the job affinity specifications.
- SubmitterType - any string identifying the type of submitter (for instance, WebUI, CLI, API, MyApp, and so on).

SubmitJobFromJSDLXmlResponse

Example 9-7 contains the type definitions for the response to the submitFromJSDLXml operation.

Example 9-7 Type definitions for response to submitJobFromJSDLXml operation

```
<xsd:element name="submitJobFromJSDLXmlResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="JobEPR" type="wsa:EndpointReferenceType" minOccurs="1"
maxOccurs="1" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

The SubmitJobFromJSDLXmlResponse consists of the job EndpointReferenceType, the EndpointReferenceType of the job just created through the submission. Any subsequent operation like cancel, get job properties, and get job output should be made on this address that contains the job ID in the reference properties.

Querying jobs

In this section we describe the operations for querying the jobs submitted to Tivoli Dynamic Workload Broker server.

The query interface accepts a certain number of filters on job attributes. Some of the filters accept wildcards such as an asterisk (*) and a question mark (?). The query operation returns the set of the jobs matching the filter criteria passed as input.

QueryJobs

Example 9-8 contains the type definitions for the queryJobs operation.

Example 9-8 Type definitions for queryJobs operation

```
<xsd:element name="queryJobs">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="NameFilter" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="TargetNamespaceFilter"
                  type="xsd:string"minOccurs="0"maxOccurs="1" />
                  <xsd:element name="AliasFilter" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="StateFilter" minOccurs="0" maxOccurs="1" >
                        <xsd:complexType>
                              <xsd:sequence>
                                    <xsd:element name="State" type="jmt:JobStateEnumeration"
                                    minOccurs="1" maxOccurs="unbounded" />
                              </xsd:sequence>
                        </xsd:complexType>
                  </xsd:element>
                        <xsd:element name="TargetResourceFilter" type="xsd:string"
                        minOccurs="0" maxOccurs="1" />

                        <xsd:element name="SubmitterFilter" type="xsd:string" minOccurs="0"
                        maxOccurs="1" />
                        <xsd:element name="SubmitTimeFilter" minOccurs="0" maxOccurs="1" >
                              <xsd:complexType>
                                    <xsd:sequence>
                                          <xsd:element name="From" type="xsd:dateTime"
                                          minOccurs="0" maxOccurs="1" />
                                          <xsd:element name="To" type="xsd:dateTime"
                                          minOccurs="0" maxOccurs="1" />
                                    </xsd:sequence>
                              </xsd:complexType>
                        </xsd:element>
                        <xsd:element name="StartTimeFilter" minOccurs="0" maxOccurs="1" >
                              <xsd:complexType><xsd:sequence>
                                    <xsd:element name="From" type="xsd:dateTime"
                                    minOccurs="0" maxOccurs="1" />
                                    <xsd:element name="To" type="xsd:dateTime" minOccurs="0"
                                    maxOccurs="1" />
                              </xsd:sequence></xsd:complexType>
                        </xsd:element>
                        <xsd:element name="EndTimeFilter" minOccurs="0" maxOccurs="1" >
```

```
                                <xsd:complexType><xsd:sequence>
                                      <xsd:element name="From" type="xsd:dateTime"
                                      minOccurs="0" maxOccurs="1" />
                                      <xsd:element name="To" type="xsd:dateTime" minOccurs="0"
                                      maxOccurs="1" />
                                </xsd:sequence></xsd:complexType>
                          </xsd:element>
                          <xsd:element name="HowMany" type="xsd:int" minOccurs="0"
                          maxOccurs="1" />
                          <xsd:element name="Iterator" type="xsd:string" minOccurs="0"
                          maxOccurs="1" />
                    </xsd:sequence>
              <xsd:anyAttribute />
        </xsd:complexType>
</xsd:element>
```

The QueryJobs consists of:

- Name filter - the filter on job names. This filter supports wildcards.
- Target Namespace Filter - the filter on job target namespaces. This filter supports wildcards. This is not supported in the Tivoli Dynamic Workload Broker V1.2 and earlier.
- Alias filter - the filter on job alias. This filter supports wildcards.
- State filter - filter on job state. It can be one of the following:
  - SUBMITTED
  - WAITING_FOR_RESOURCES
  - RESOURCE_ALLOCATION_RECEIVED
  - RESOURCE_ALLOCATION_FAILED
  - SUBMITTED_TO_ENDPOINT
  - PENDING_CANCEL
  - CANCEL_ALLOCATION
  - RESOURCE_REALLOCATE
  - EXECUTING
  - FAILED_EXECUTION
  - SUCCEEDED_EXECUTION
  - NOT_EXECUTED
  - CANCELLED
  - UNKNOWN
- Target Resource Filter - This filter matches all jobs executed on the specified resource display name.
- Submitter Filter - This filter matches all job submitted by the specified submitter. This filter supports wildcards.

- Submit time filter - This filter matches all jobs submitted to Tivoli Dynamic Workload Broker server in the specified range.
- Start time filter - This filter matches all jobs started on the endpoint in the specified range.
- End time filter - This filter matches all jobs ended on the endpoint in the specified range.
- HowMany - This is the number of maximum jobs that the query should return in one invocation. Note that if the number of jobs matching the filter criteria is greater that this number then an iterator is returned and it can be used to retrieve all other jobs not returned.
- Iterator - This is an identifier used by the Tivoli Dynamic Workload Broker server to query the remaining jobs matching the criteria and exceeding the HowMany parameter.

QueryJobsProperties

Example 9-9 describes all job attributes returned by the query operation.

Example 9-9 Type definitions for queryJobProperties - response to queryJobs operation

```
<xsd:complexType  name="QueryJobPropertiesType">
      <xsd:sequence>
            <xsd:element name="JobEPR" type="wsa:EndpointReferenceType" minOccurs="1"
            maxOccurs="1" />
            <xsd:element name="Name" type="xsd:QName" minOccurs="1" maxOccurs="1" />
            <xsd:element name="Alias" type="xsd:string" minOccurs="0" maxOccurs="1" />
            <xsd:element name="Submitter" type="xsd:string" minOccurs="0" maxOccurs="1" />
            <xsd:element name="SubmitterType" type="xsd:string" minOccurs="0" maxOccurs="1" />
            <xsd:element name="State" type="jmt:JobStateEnumeration" minOccurs="1"
            maxOccurs="1" />
            <xsd:element name="LastStatusMessage" type="xsd:string" minOccurs="0"
            maxOccurs="1" />
            <xsd:element name="SubmittedTime" type="xsd:dateTime" minOccurs="0"
            maxOccurs="1" />
            <xsd:element name="StartTime" type="xsd:dateTime" minOccurs="0" maxOccurs="1" />
            <xsd:element name="EndTime" type="xsd:dateTime" minOccurs="0" maxOccurs="1" />
            <xsd:element name="Duration" type="xsd:duration" minOccurs="0" maxOccurs="1" />
            <xsd:element name="ReturnCode" type="xsd:integer" minOccurs="0" maxOccurs="1" />
            <xsd:element name="TargetResource" type="rmt:ResourceGroup" minOccurs="0"
            maxOccurs="unbounded" />
      </xsd:sequence>
</xsd:complexType>
```

The QueryJobsProperties consists of:

- Name - the job name.
- Alias - the job alias.
- Submitter - the job submitter.
- Submitter type - the job submitter type.
- State - job state.
- Last status message - the last diagnostic message received for the job.
- Submitted time - job submit time.
- Start time - the job start time.
- End time - the job end time.
- Duration - the duration of job from start to end time.
- Return code - the job return code. Valid only for native jobs.
- Target resource - the target group of resources matching the requirements.

QueryJobsResponse

Example 9-10 contains the type definitions for all parameters returned by the query operation. The QueryJobsResponse consists of:

- Job Properties - the set of jobs matching the filter criteria
- Iterator - the iterator to be used in case there are more jobs matching the requirements than requested with HowMany

Example 9-10 Type definitions for response to queryJobProperties operation

```
<xsd:element name="queryJobsResponse">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="JobProperties" type="jmjf:QueryJobPropertiesType"
                  minOccurs="0" maxOccurs="unbounded" />
                  <xsd:element name="Iterator" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
            </xsd:sequence>
            <xsd:anyAttribute />
      </xsd:complexType>
</xsd:element>
```

9.4.3 Job service

In this section we describe the job service. This service provides the following functionality:

- Cancelling jobs
- Getting job properties
- Getting the job output

Example 9-11 contains the extract from the JobFactory.wsdl file. You can see all of the operations defined with this Web Service.

Example 9-11 Job Web service and its operations for submitting and querying jobs

```
<wsdl:portType name="Job">

   <wsdl:operation name="cancel">
      <wsdl:input name="CancelRequest" message="jmj:CancelRequestMessage" />
      <wsdl:output name="CancelResponse" message="jmj:CancelResponseMessage" />
      <wsdl:fault name="OperationFailedFault" message="jmj:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jmj:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="UnknownResourceFault" message="jmj:UnknownResourceFaultMessage" />
      <wsdl:fault name="IllegalStateFault" message="jmj:IllegalStateFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jmj:ServiceUnavailableFaultMessage"
/>
      <wsdl:fault name="UnsupportedOperationFault"
message="jmj:UnsupportedOperationFaultMessage" />
   </wsdl:operation>

   <wsdl:operation name="getProperties">
      <wsdl:input name="GetPropertiesRequest" message="jmj:GetPropertiesRequestMessage" />
      <wsdl:output name="GetPropertiesResponse" message="jmj:GetPropertiesResponseMessage" />
      <wsdl:fault name="OperationFailedFault" message="jmj:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jmj:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="UnknownResourceFault" message="jmj:UnknownResourceFaultMessage" />
      <wsdl:fault name="IllegalStateFault" message="jmj:IllegalStateFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jmj:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

   <wsdl:operation name="getExecutionLogPage">
      <wsdl:input name="GetExecutionLogPageRequest"
message="jmj:GetExecutionLogPageRequestMessage" />
      <wsdl:output name="GetExecutionLogPageResponse
"message="jmj:GetExecutionLogPageResponseMessage" />
      <wsdl:fault name="OperationFailedFault" message="jmj:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jmj:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="UnknownResourceFault" message="jmj:UnknownResourceFaultMessage" />
      <wsdl:fault name="IllegalStateFault" message="jmj:IllegalStateFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jmj:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

</wsdl:portType>


<wsdl:service name="JobFactoryService">
      <wsdl:port binding="jmjf:JobFactoryBinding" name="JobFactory">
            <wsdlsoap:address
            location="http://localhost:9080/JobServiceWS/services/JobFactory" />
      </wsdl:port>
</wsdl:service>
```

Canceling jobs

In this section we describe the operation for cancelling the jobs submitted to Tivoli Dynamic Workload Broker server.

The cancel interface allows us to stop the execution of a running job or to block the process of allocating the matching resources for a job that was not launched yet. This operation is asynchronous: it just verifies if the job identifier exists and then sends a cancel request to the appropriate Tivoli Dynamic Workload Broker component.

The cancel operation must be invoked on the address contained in the Job EndpointReferenceType (EPR) returned back from the submit operation (submitJobFromName, submitJobFromJSDL, submitJobFromJSDLXml). The job EPR also contains the job identifier as a reference property named "jobIdentifier" that should set in the SOAP header when invoking this operation.

Cancel

As stated above, it is important that you invoke this operation on the address contained in the job's EPR. Additional parameters are not needed. When you invoke nthe cancel operation in your application, you must invoke it on the correct EPR instance, and you do not provide the cancel method with any argument.

This is the reason that we do not provide type definitions for the cancel operation.

There is also no response to the cancel operation. This is the reason that we do not provide type definitions for response to the cancel operation.

Getting job properties

In this section we describe the operation for getting the properties of jobs submitted to the Tivoli Dynamic Workload Broker server.

The getProperties interface allows you to get the properties of a submitted job. The getProperties operation must be invoked on the address in the job EndpointReferenceType (EPR) returned back from jobSubmit. The job EPR also contains the job identifier as reference property named “jobIdentifier” that should be set in the SOAP header when invoking this operation.

GetProperties

As stated above, it is important for you to invoke this operation on the address contained in the job’s EPR. Additional parameters are not needed. When you invoke the getProperties operation in your application, you must invoke it on the correct EPR instance, and you will not provide the getProperties method with any argument.

This is the reason that we do not provide type definitions for the getProperites operation.

GetPropertiesResponse

Example 9-12 contains the type definitions for the queryJobs operation.

Example 9-12 Type definitions for response to getProperties operation

```
<xsd:element name="getPropertiesResponse">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="Name" type="xsd:QName" minOccurs="1" maxOccurs="1" />
                  <xsd:element name="Alias" type="xsd:string" minOccurs="0" maxOccurs="1" />
                  <xsd:element name="Submitter" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="SubmitterType" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="ClientNotifyEPR" type="wsa:EndpointReferenceType"
                  minOccurs="0"   maxOccurs="1" />
                  <xsd:element name="JobDefinitionDocument" minOccurs="1" maxOccurs="1">
                        <xsd:complexType><xsd:sequence>
                              <xsd:element ref="jsdl:jobDefinition" minOccurs="1"
                              maxOccurs="1" />
                        </xsd:sequence></xsd:complexType>
                  </xsd:element>
                  <xsd:element name="State" type="jmt:JobStateEnumeration" minOccurs="1"
                  maxOccurs="1" />
                  <xsd:element name="LastStatusMessage" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="SubmittedTime" type="xsd:dateTime" minOccurs="1"
                  maxOccurs="1" />
                  <xsd:element name="StartTime" type="xsd:dateTime" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="EndTime" type="xsd:dateTime" minOccurs="0"
                  maxOccurs="1" />
```

```
                <xsd:element name="Duration" type="xsd:duration" minOccurs="0"
                maxOccurs="1" />
                <xsd:element name="ReturnCode" nillable="true" type="xsd:integer"
                minOccurs="0" maxOccurs="1" />
                <xsd:element name="TargetResource" type="rmt:ResourceGroup" minOccurs="0"
                maxOccurs="unbounded" />
                <xsd:element name="Metric" type="jmt:JobUsageMetricsType" minOccurs="0"
                maxOccurs="unbounded" />
            </xsd:sequence>
            <xsd:anyAttribute />
        </xsd:complexType>
</xsd:element>
```

The GetPropertiesResponse consists of:

- Name - the job name.
- Alias - the job alias.
- Submitter - the job submitter.
- Submitter type - the job submitter type.
- Client notification EndpointReferenceType - the address of the client implementing the notify Web service passed in the submitJob.
- Job definition document - the JSDL passed in the submitJob with all variables replaced.
- State - the job state.
- Last status message - the last diagnostic message received for the job.
- Submitted time - the job submit time.
- Start time - the job start time.
- End time - the job end time.
- Duration - the duration of the job from start to end time.
- Return code - the job return code. This is valid only for native.
- Target resource - the target group of resources matching the requirements.
- Metric - a set of metrics (CPU used and Max memory) about resource consumed by the native jobs.

Getting the job output

In this section we describe the operation for getting the output of jobs submitted to the Tivoli Dynamic Workload Broker server.

The getExecutionLogPage operation gets the execution log from the endpoint where the job has been dispatched. The getExecutionLogPage operation must be invoked on the address contained in the Job EndpointReferenceType (EPR) returned back from jobSubmit. The job EPR also contains the job identifier as a reference property named "jobIdentifier" that should be set in the SOAP header when invoking this operation.

GetExecutionLogPage

Example 9-13 contains the type definitions for the getExecutionLogPage operation.

Example 9-13 Type definitions for getExecutionLogPage operation

```
<xsd:element name="getExecutionLogPage">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="Offset" type="xsd:long" minOccurs="1" maxOccurs="1" />
                  <xsd:element name="BlockSize" type="xsd:int" minOccurs="1" maxOccurs="1" />
            </xsd:sequence>
            <xsd:anyAttribute />
      </xsd:complexType>
</xsd:element>
```

The GetExecutionLogPage type consists of:

- Offset - the starting offset of the requested page. For the first page 0 should be specified. If the log to be returned is greater than the requested BlockSize then the operation must be called as many times as necessary until the offset goes over the file size. Any subsequent call should specify the offset on based BlockSize and FileSize.
- BlockSize - the size in bytes of the requested log page.

GetExecutionLogPageResponse

Example 9-14 contains the type definitions for the getExecutionLogPage operation.

Example 9-14 Type definitions for response to getExecutionLogPage operation

```
<xsd:element name="getExecutionLogPageResponse">
      <xsd:complexType>
         <xsd:sequence>
               <xsd:element name="LogPage" type="xsd:string" minOccurs="0" maxOccurs="1" />
               <xsd:element name="FileSize" type="xsd:long" minOccurs="1" maxOccurs="1" />
         </xsd:sequence>
         <xsd:anyAttribute />
      </xsd:complexType>
</xsd:element>
```

The GetExecutionLogPage type consists of:

- LogPage: the page starting from the requested offset and with the given block size.
- FileSize: the size in bytes of the entire log. It may differ call by call, because the job may produce more rows in the meantime.

9.4.4 Job Definition Management service

In this section we describe the Job Definition Management service. This service provides the following functionality:

- Adding the job definitions
- Modifying the job definitions
- Deleting the job definitions
- Querying the job definitions

Example 9-15 contains the extract from the JobDefinitionMgmt.wsdl file. You can see all of the operations defined with this Web service.

Example 9-15 Job Definition Management Service and its operations for working with job definitions

```
<wsdl:portType name="JobDefinitionManagement">

   <wsdl:operation name="addJobDefinition">
      <wsdl:input name="AddJobDefinitionRequest" message="jdm:AddJobDefinitionRequestMessage"
/>
      <wsdl:output name="AddJobDefinitionResponse"
message="jdm:AddJobDefinitionResponseMessage" />
      <wsdl:fault name="OperationFailedFault" message="jdm:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jdm:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jdm:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

   <wsdl:operation name="addJobDefinitionXml">
      <wsdl:input name="AddJobDefinitionXmlRequest"
message="jdm:AddJobDefinitionXmlRequestMessage" />
      <wsdl:output name="AddJobDefinitionXmlResponse"
message="jdm:AddJobDefinitionXmlResponseMessage" />
      <wsdl:fault name="OperationFailedFault" message="jdm:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jdm:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jdm:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

   <wsdl:operation name="setJobDefinition">
```

```
      <wsdl:input name="SetJobDefinitionRequest" message="jdm:SetJobDefinitionRequestMessage"
/>
      <wsdl:output name="SetJobDefinitionResponse"
message="jdm:SetJobDefinitionResponseMessage" />
      <wsdl:fault name="OperationFailedFault" message="jdm:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jdm:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jdm:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

   <wsdl:operation name="setJobDefinitionXml">
      <wsdl:input name="SetJobDefinitionXmlRequest"
message="jdm:SetJobDefinitionXmlRequestMessage" />
      <wsdl:output name="SetJobDefinitionXmlResponse"
message="jdm:SetJobDefinitionXmlResponseMessage" />
      <wsdl:fault name="OperationFailedFault" message="jdm:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jdm:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jdm:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

   <wsdl:operation name="getJobDefinition">
      <wsdl:input name="GetJobDefinitionRequest" message="jdm:GetJobDefinitionRequestMessage"
/>
      <wsdl:output name="GetJobDefinitionResponse"
message="jdm:GetJobDefinitionResponseMessage" />
      <wsdl:fault name="UnknownResourceFault" message="jdm:UnknownResourceFaultMessage" />
      <wsdl:fault name="OperationFailedFault" message="jdm:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jdm:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jdm:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

   <wsdl:operation name="getJobDefinitionXml">
      <wsdl:input name="GetJobDefinitionXmlRequest"
message="jdm:GetJobDefinitionXmlRequestMessage" />
      <wsdl:output name="GetJobDefinitionXmlResponse"
message="jdm:GetJobDefinitionXmlResponseMessage" />
      <wsdl:fault name="UnknownResourceFault" message="jdm:UnknownResourceFaultMessage" />
      <wsdl:fault name="OperationFailedFault" message="jdm:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jdm:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jdm:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

   <wsdl:operation name="deleteJobDefinition">
      <wsdl:input name="DeleteJobDefinitionRequest"
message="jdm:DeleteJobDefinitionRequestMessage" />
```

```
      <wsdl:output name="DeleteJobDefinitionResponse"
message="jdm:DeleteJobDefinitionResponseMessage" />
      <wsdl:fault name="UnknownResourceFault" message="jdm:UnknownResourceFaultMessage" />
      <wsdl:fault name="OperationFailedFault" message="jdm:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jdm:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jdm:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

   <wsdl:operation name="queryJobDefinitions">
      <wsdl:input name="QueryJobDefinitionsRequest"
message="jdm:QueryJobDefinitionsRequestMessage" />
      <wsdl:output name="QueryJobDefinitionsResponse"
message="jdm:QueryJobDefinitionsResponseMessage" />
      <wsdl:fault name="OperationFailedFault" message="jdm:OperationFailedFaultMessage" />
      <wsdl:fault name="InvalidArgumentsFault" message="jdm:InvalidArgumentsFaultMessage" />
      <wsdl:fault name="ServiceUnavailableFault" message="jdm:ServiceUnavailableFaultMessage"
/>
   </wsdl:operation>

</wsdl:portType>

<wsdl:service name="JobDefinitionManagementService">
   <wsdl:port binding="jdm:JobDefinitionManagementBinding" name="JobDefinitionManagement">
      <wsdlsoap:address location="http://localhost:9080/JobStoreWS/services/JobStore" />
   </wsdl:port>
</wsdl:service>
```

Adding the job definitions

In this section we describe the operations for adding the job definitions.

The addJobDefinition operation saves a given job definition in the server job repository. Saved job definitions can be referenced using the submitJobFromName operation.

AddJobDefinition

Example 9-16 contains the type definitions for the addJobDefinition operation.

Example 9-16 Type definitions for addJobDefinition operation

```
<xsd:element name="addJobDefinition">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:elementname="JobDefinition"type="
                  jsdl:JobDefinitionType" minOccurs="1" maxOccurs="1" />
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

The AddJobDefinition type consists of JobDefinition, the job definition to be saved.

AddJobDefinitionResponse

Example 9-17 contains the type definitions for the response to the addJobDefinition operation.

Example 9-17 Type definitions for response to addJobDefinition operation

```
<xsd:element name="addJobDefinitionResponse">
      <xsd:complexType>
            <xsd:sequence />
      </xsd:complexType>
</xsd:element>
```

Updating the job definitions

In this section we describe the operations for updating the job definitions.

The setJobDefinition operation updates a given job definition in the server job repository.

SetJobDefinition

Example 9-18 contains the type definitions for the setJobDefinition operation.

This operation can be used only for updating existing job definitions.

Example 9-18 Type definitions for setJobDefinition operation

```
<xsd:element name="setJobDefinition">
      <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="JobDefinition"type="jsdl:JobDefinitionType"
               minOccurs="1" maxOccurs="1" />
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

The SetJobDefinition type consists of JobDefinition, the job definition to be saved.

SetJobDefinitionResponse

Example 9-19 contains the type definitions for the response to the setJobDefinition operation.

Example 9-19 Type definitions for response to setJobDefinition operation

```
<xsd:element name="setJobDefinitionResponse">
      <xsd:complexType>
            <xsd:sequence />
      </xsd:complexType>
</xsd:element>
```

Deleting the job definitions

In this section we describe the operations for deleting the job definitions.

The deleteJobDefinition operation deletes a given job definition in the server job repository.

DeleteJobDefinition

Example 9-20 contains the type definitions for the deleteJobDefinition operation.

This operation deletes existing job definitions.

Example 9-20 Type definitions for deleteJobDefinition operation

```
<xsd:element name="deleteJobDefinition">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="Name" type="xsd:QName" minOccurs="1" maxOccurs="1" />
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

<xsd:element name="deleteJobDefinitionResponse">
      <xsd:complexType>
            <xsd:sequence />
      </xsd:complexType>
</xsd:element>
```

The DeleteJobDefinition type consists of Name, the name of the JobDefinition (for instance, the job name) to be deleted. It is a QName, then it is of the form "namespace:jobname". Nevertheless, namespaces different from the empty strings are not supported, so it should be the job name only.

Querying the job definitions

In this section we describe the operations for querying the job definitions.

The queryJobDefinitions operation queries the job definitions stored in the server job repository.

QueryJobDefinitions

This type describes all parameters passed to the queryJobDefinition operation.

This operation queries existing job definitions.

Example 9-21 contains the type definitions for the queryJobDefinitions operation.

Example 9-21 Type definitions for queryJobDefinitions operation

```
<xsd:element name="queryJobDefinitions">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="NameFilter" type="xsd:string" minOccurs="0"
                  maxOccurs="1"  />
                  <xsd:element name="TargetNamespaceFilter" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="DescriptionFilter" type="xsd:string" minOccurs="0"
                  maxOccurs="1"  />
                  <xsd:element name="UserFilter" type="xsd:string" minOccurs="0"
                  maxOccurs="1"  />
                  <xsd:element name="HowMany" type="xsd:int" minOccurs="0" maxOccurs="1" />
                  <xsd:element name="Iterator" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

The QueryJobDefinition type consists of:

- Name Filter - the filter on job names. This filter supports wildcards.
- Target Namespace Filter - the filter on job target namespaces. This filter supports wildcards. This is *not* Supported for NOW™.
- Description Filter - the filter on job descriptions. This filter supports wildcards.
- User Filter - This filter matches all job created by the specified user. This filter supports wildcards.
- HowMany - the number of maximum job definitions that the query should return in one invocation. Note that if the number of job definitions matching the filter criteria is greater than this number then an iterator is returned and it can be used to retrieve all other jobs not returned back.
- Iterator - an identifier used by the Tivoli Dynamic Workload Broker server to query the remaining job definitions matching the criteria and exceeding the HowMany parameter.

QueryJobDefinitionProperties

Example 9-22 contains the type definitions describing all job definition attributes returned by the query operation.

Example 9-22 Type definitions for QueryJobDefinitionPropertiesType

```
<xsd:complexType name="QueryJobDefinitionPropertiesType">
      <xsd:sequence>
            <xsd:element name="Name" type="xsd:QName" minOccurs="1" maxOccurs="1" />
            <xsd:element name="Description" type="xsd:string" minOccurs="1" maxOccurs="1" />
            <xsd:element name="Owner" nillable="true" type="xsd:string" minOccurs="1"
            maxOccurs="1" />
            <xsd:element name="CreationTime" type="xsd:dateTime" minOccurs="1"
            maxOccurs="1" />
            <xsd:element name="ModificationTime" type="xsd:dateTime" minOccurs="1"
            maxOccurs="1" />
      </xsd:sequence>
</xsd:complexType>
```

The QueryJobDefinitionProperties consists of:

- Name - the job definition name.
- Description - the job definition description.
- Owner - the job definition creator.
- Creation Time - the job definition creation time.
- Modification Time - the job definition modification time.

QueryJobDefinitionsResponse

Example 9-23 contains the type definitions describing all parameters returned by the query operation.

Example 9-23 Type definitions for response to query operation

```
<xsd:element name="queryJobDefinitionsResponse">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="JobDefinitionProperties"
                  type="jdm:QueryJobDefinitionPropertiesType" minOccurs="0"
                  maxOccurs="unbounded" />
                  <xsd:element name="Iterator" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

The QueryJobDefinitionsResponse consists of:

- Job Properties - the set of job definitions matching the filter criteria
- Iterator - the iterator to be used in case there are more job definitions matching the requirements than requested with HowMany

Getting the job definition

In this section we describe the operations for getting the job definition.

The getJobDefinition interface gets the job definition in the server job repository.

GetJobDefinition

Example 9-24 contains the type definitions for the getJobDefinition operation. This operation gets an existing job definition.

Example 9-24 Type definitions for getJobDefinition operation

```
<xsd:element name="getJobDefinition">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="Name" type="xsd:QName" minOccurs="1" maxOccurs="1" />
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

The GetJobDefinition type consists of Name, the name of the requested job definition.

GetJobDefinitionResponse

Example 9-25 contains the type definitions for the response to the getJobDefinitionResponse operation.

Example 9-25 Type definitions for response to getJobDefinition operation

```
<xsd:element name="getJobDefinitionResponse">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="JobDefinition" type="jsdl:JobDefinitionType"
                  minOccurs="1" maxOccurs="1" />
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

The GetJobDefinitionResponse type consists of Job Definition, the requested job definition.

9.4.5 Important terms related to job definitions

In this section we briefly introduce two important terms related to job submission:

- Variable substitution - Default values contained in a job's definition can be overriden at submission time.
- Job Affinity - A job affine to another should be dispatched.

Understanding this section in detail is not necessary if you are going through this chapter for the first time. However, it may be useful at later time.

Variables substitution

Example 9-26 describes the type for passing of *variables* to the job at submission time. The variables passed at submit time override the default value already defined within the JSDL.

A variable consists of:

- Name: It can be referenced where possible in the JSDL using $ {var_name } notation.
- Value: The actual value of the variable. It is converted to the correct type when the variable substitution occurs at submission time.

Example 9-26 Type definitions for job variables

```
<xsd:complexType name="VariableType">
      <xsd:sequence>
            <xsd:element name="Name" type="xsd:NCName" minOccurs="1" maxOccurs="1" />
            <xsd:element name="Value" type="xsd:string" minOccurs="1" maxOccurs="1" />
      </xsd:sequence>
</xsd:complexType>
```

Job affinity

Example 9-27 describes the type for *affinity* constraints. There are three possible job affinity types:

- Job EndpointReferenceType: This contains the ID of the job to which the submitting job should be affine. The Tivoli Dynamic Workload Broker dispatches the submitting job to the same target as the affine one.
- Alias: The alias of the job to which the submitting job should be affine. The Tivoli Dynamic Workload Broker server dispatches the submitting job to the same target as the affine one.
- Resource group: The resource group containing the target resource where the submitting job should be dispatched.

Example 9-27 Type definitions for job affinity

```
<xsd:complexType name="AffinityType">
         <xsd:choice>
               <xsd:element name="JobEPR" type="wsa:EndpointReferenceType" minOccurs="1"
               maxOccurs="1" />
               <xsd:element name="Alias" type="xsd:string" minOccurs="1" maxOccurs="1" />
               <xsd:element name="Resources" type="rmt:ResourceGroup"  minOccurs="0"
               maxOccurs="unbounded" />
         </xsd:choice>
</xsd:complexType>
```

For more information about variable substitution and job affinity, refer to *IBM Tivoli Dynamic Workload Broker User's Guide Version 1.1*, SC32-2281.

9.4.6 Getting notified about job state changes

In this section we describe how the Tivoli Dynamic Workload Broker can be notified about the job state changes.

Tivoli Dynamic Workload Broker clients submitting a job through the Web service interfaces can be notified about job state changes if they implement the NotificationConsumer service. As described above, it is possible to pass an EndpointReferenceType in the submit that represents the address of the NotificationConsumer service.

JobStateEnumeration

Example 9-28 contains type definitions describing all possible job states.

Example 9-28 Type definitions for JobStateEnumeration

```
<xsd:simpleType name="JobStateEnumeration">
      <xsd:annotation>
            <xsd:documentation>
                  States for a Job.
            </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:string">
            <xsd:enumeration value="SUBMITTED" />
            <xsd:enumeration value="WAITING_FOR_RESOURCES" />
            <xsd:enumeration value="RESOURCE_ALLOCATION_RECEIVED" />
            <xsd:enumeration value="RESOURCE_ALLOCATION_FAILED" />
            <xsd:enumeration value="SUBMITTED_TO_ENDPOINT" />
            <xsd:enumeration value="PENDING_CANCEL" />
            <xsd:enumeration value="CANCEL_ALLOCATION" />
            <xsd:enumeration value="RESOURCE_REALLOCATE" />
            <xsd:enumeration value="EXECUTING" />
            <xsd:enumeration value="FAILED_EXECUTION" />
            <xsd:enumeration value="SUCCEEDED_EXECUTION" />
            <xsd:enumeration value="NOT_EXECUTED" />
            <xsd:enumeration value="CANCELLED" />
            <xsd:enumeration value="UNKNOWN" />
      </xsd:restriction>
</xsd:simpleType>
```

JobUsageMetricsType

Example 9-29 contains the type definitions describing the metrics associated to each job. These are expressed with a set of name-value-type triples. The possible names are:

- "JOB_CPU_USAGE" with type "DECIMAL"
- "JOB_MEMORY_USAGE" with type "DECIMAL"

Example 9-29 Type definitions for metrics associated with jobs

```
<xsd:complexType name="JobUsageMetricsType">
      <xsd:sequence>
            <xsd:element name="Name" type="xsd:NCName" minOccurs="1"
            maxOccurs="1" />
            <xsd:element name="Value" type="xsd:string" minOccurs="1"
            maxOccurs="1" />
            <xsd:element name="Type" type="xsd:string" minOccurs="0"
            maxOccurs="1" />
      </xsd:sequence>
</xsd:complexType>
```

NotifyJobStatusChange

Example 9-30 contains the type definitions describing the job attributes sent in the notification.

Example 9-30 Type definitions for the job attributes sent in the notification

```
<xsd:element name="NotifyJobStatusChange">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="State" type="jmt:JobStateEnumeration" maxOccurs="1"
                  minOccurs="1" />
                  <xsd:element name="Alias" type="xsd:string" minOccurs="0" maxOccurs="1" />
                  <xsd:element name="StatusMessage" type="xsd:string" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="SubmittedTime" type="xsd:dateTime" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="StartTime" type="xsd:dateTime" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="EndTime" type="xsd:dateTime" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="Duration" type="xsd:duration" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="ReturnCode" type="xsd:integer" minOccurs="0"
                  maxOccurs="1" />
                  <xsd:element name="Metric" type="jmt:JobUsageMetricsType" minOccurs="0"
                  maxOccurs="unbounded" />
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

The NotifyJobStatusChange consists of:

- Alias - the job alias.
- State - the job state.
- Status Message - the last diagnostic message received for the job.
- Submitted Time - the job submit time.
- Start Time - the job start time.
- End Time - the job end time.
- Duration - the duration of the job from start to end time.
- Return code - the job return code. This is valid only for native.
- Metric - a set of metrics (CPU used and Max memory) about resource consumed by the native jobs.

NotificationMessageHolderType

Example 9-31 contains the type definitions describing the otification message passed in the notify operation.

Example 9-31 Type definitions for notification message

```
<xsd:complexType name="NotificationMessageHolderType">
      <xsd:sequence>
            <xsd:element name="Topic" type="xsd:string" maxOccurs="1" minOccurs="1" />
            <xsd:element name="ProducerReference" type="wsa:EndpointReferenceType"
            maxOccurs="1" minOccurs="1" />
            <xsd:element name="ProducerID" type="xsd:string" maxOccurs="1" minOccurs="1" />
            <xsd:element name="Message" type="xsd:anyType" maxOccurs="1" minOccurs="1"/>
      </xsd:sequence>
</xsd:complexType>
```

The NotificationMessageHolderType is made of the following elements:

- Topic - It can only be "JobStatusChange".
- Producer Reference - the EPR of the job changing the state.
- Producer ID - not used.
- Message - contains the relevant information about the job. The actual type is NotifyJobStatusChange.

9.5 Creating the sample client

In this section we provide a step-by-step explanation of the process of building the client application leveraging the Web services interface provided by the Tivoli Dynamic Workload Broker server.

We create a sample client that will be able to submit a job from the JSDL definition stored in the file. The client application monitors the status of the submitted job until it ends, either successfully or not.

We describe the following topics:

- Development environments used in our scenarios
- Steps for creating the project in Rational Application Developer:
  - Creating a project with defined input/output paths
  - Including the directory with Web services related files into your project
  - Generating the Web Service client Java packages from the WSDL file
  - Adding the application logic and linking it together with the Web Service client Java packages
  - Running the client within the Rational Application Developer environment
- Steps for creating the project using Eclipse together with utilities and JAR files provided with WebSphere Application Server V6.1:
  - Creating a project with defined input/output paths
  - Creating the Web Service client (generating the Web Service client Java packages from the WSDL definition using the WSDL2Java command-line utility)
  - Providing the generated Web Service client Java packages created in previous step to the project
  - Adding the application logic and linking it together with the Web service client Java packages
  - Running the project within Eclipse
- Running the client from the command line
- Java run times and JAR files necessary for running the client from the command line

Note: From the lists shown above it may seem that the steps for creating the sample client are almost the same. They are similar, but they differ in a few important steps. In the Rational Application Developer scenario we do not directly call any command-line utility, while in the Eclipse scenario we call the WSDL2Java utility directly from the command line. Also, the ways in which we provide the Web Service client Java packages are different for Rational Application Developer and Eclipse.

9.5.1 Development tools used in our scenarios

There are many Integrated Development Environments (IDEs) that allow you to develop applications based on a Java platform. In the following sections we demonstrate how to build the sample client using two Integrated Development Environments:

- Rational® Application Developer V7.0
- Eclipse in conjunction with libraries and tools provided with WebSphere Application Server V6.1

9.5.2 Creating the sample client using Rational Application Developer

In this section we describe how to create a sample client application leveraging the Tivoli Dynamic Workload Broker Web services interface. We use the Rational Application Developer V7.0 for this task.

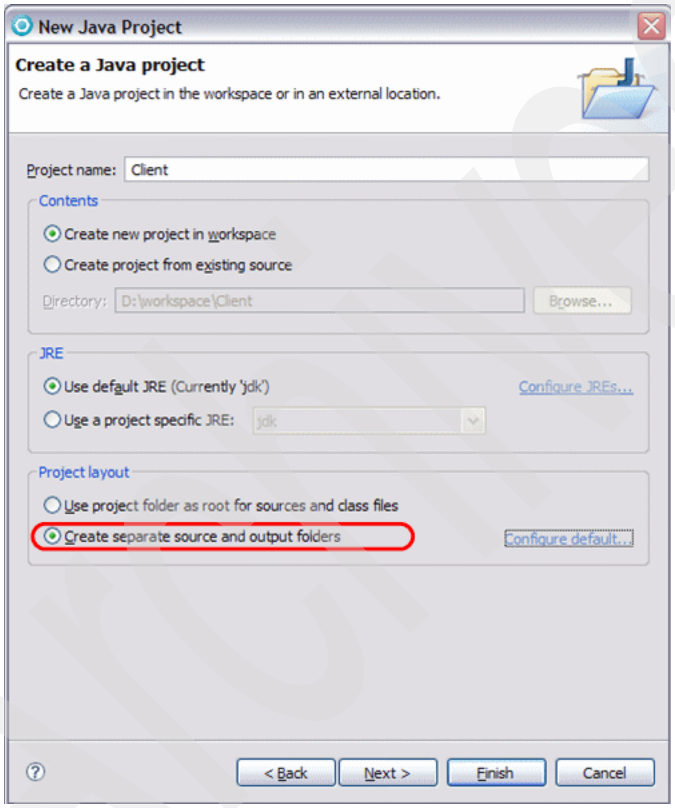
We describe the actions that must be taken in order to successfully create the sample client from scratch. The step-by-step instructions describe the following tasks:

- Creating the project
- Providing the WSDL definitions to the project
- Creating the Web Service client (generating Java packages from the WSDL definition)
- Adding the application logic and linking it together with the Java packages created in previous step
- Running the project

Creating the project

In this section we demonstrate how to create the project for our sample client.

1. Launch the Rational Application Developer and create a new project by selecting **File** → **New Project**. Select **Java project** and click **Next**.
2. In the following window, type in the project name. Type `Client` into the Project Name text box. Then specify separate subdirectories for the source code and for the output (Java classes). Select **Create separate source and output folders** (Figure 9-3).



New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: Client

Contents
- [✓ Create new project in workspace]
- Create project from existing source

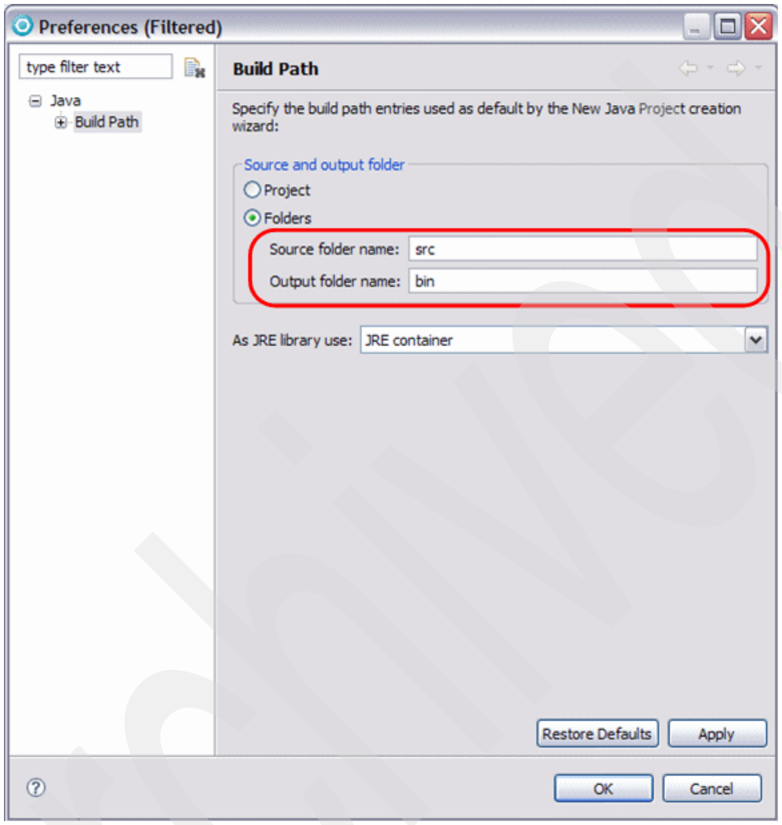
Directory: D:\workspace\Client

JRE
- [✓ Use default JRE (Currently 'jdk')]
- Use a project specific JRE: jdk

Project layout
- Use project folder as root for sources and class files
- [✓ Create separate source and output folders]

Figure 9-3 Creating a new Java Project

3. To make sure which directories are dedicated for the input (sources) and output (classes), click **Configure default**, and in the following window check the values of the source folder name and output folder name (Figure 9-4).



Specify the build path entries used as default by the New Java Project creation wizard:

Source and output folder

Project

[✓ Folders]

Source folder name: src

Output folder name: bin

As JRE library use: JRE container

Figure 9-4 Selecting the source and output folders

4. Keep the following values:
   - Source folder name: src
   - Output folder name: bin
5. Click **OK** to close the Preferences window and then click **Next** and **Finish** in the following window.

Based on your configuration, you should have your workspace divided into several panes. If you do not see the Package Explorer pane in your workspace, switch the view by selecting **Window** → **Show view** → **Package Explorer** (Figure 9-5).

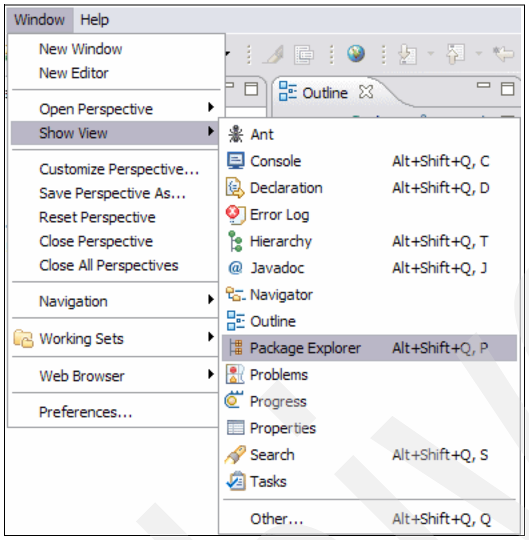
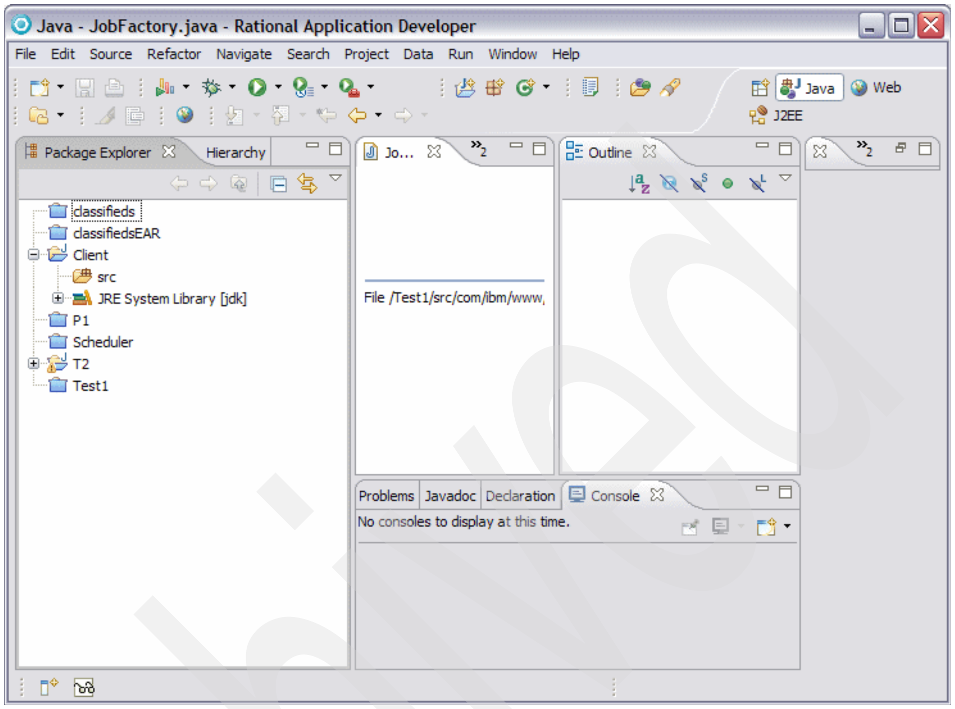


Figure 9-5 Switching to Package Explorer View

Now your workspace should look like Figure 9-6.



File /Test1/src/com/ibm/www,

No consoles to display at this time.

Figure 9-6 Rational Application Developer - workspace

Providing the WSDL files to the project

In this section we demonstrate how to provide the WSDL files located on the installation media to the project.

1. Create the additional directory within the project structure. This directory contains the WSDL definitions and other related files.

2. In the Package Explorer pane right-click our client project. Select **New** → **Folder** (Figure 9-7).

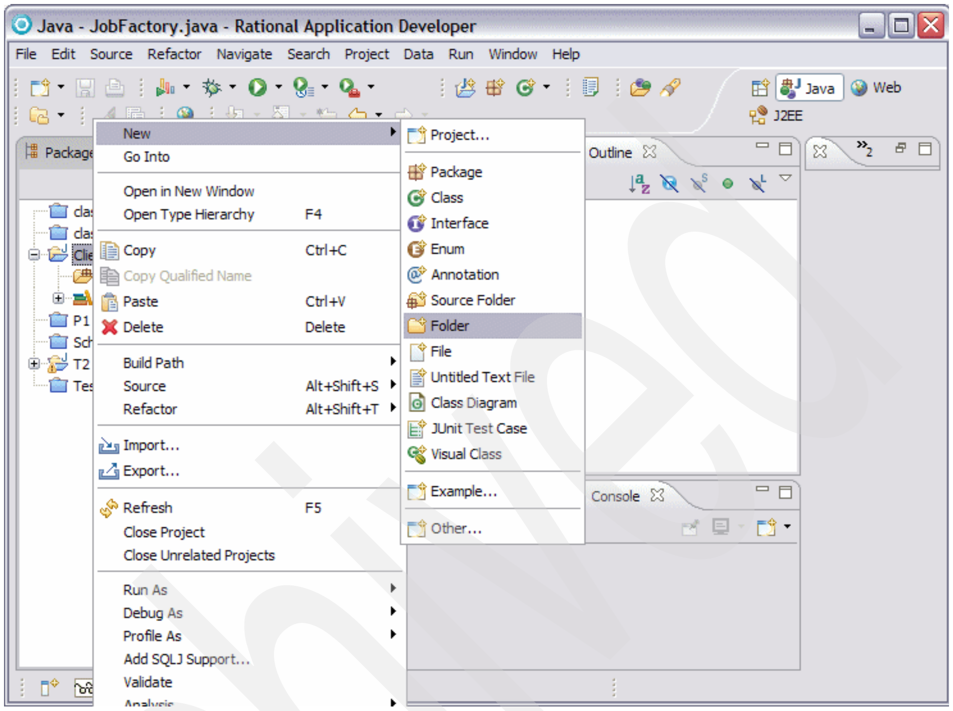


Figure 9-7 Adding a folder to the project - 1

3. In the following window, type the folder name. Type `wsdl` into the Folder name text box (Figure 9-8).

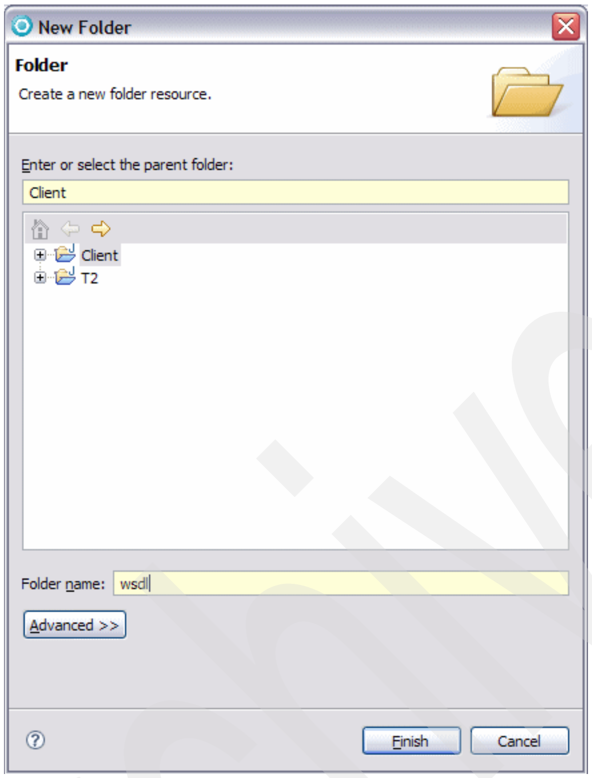


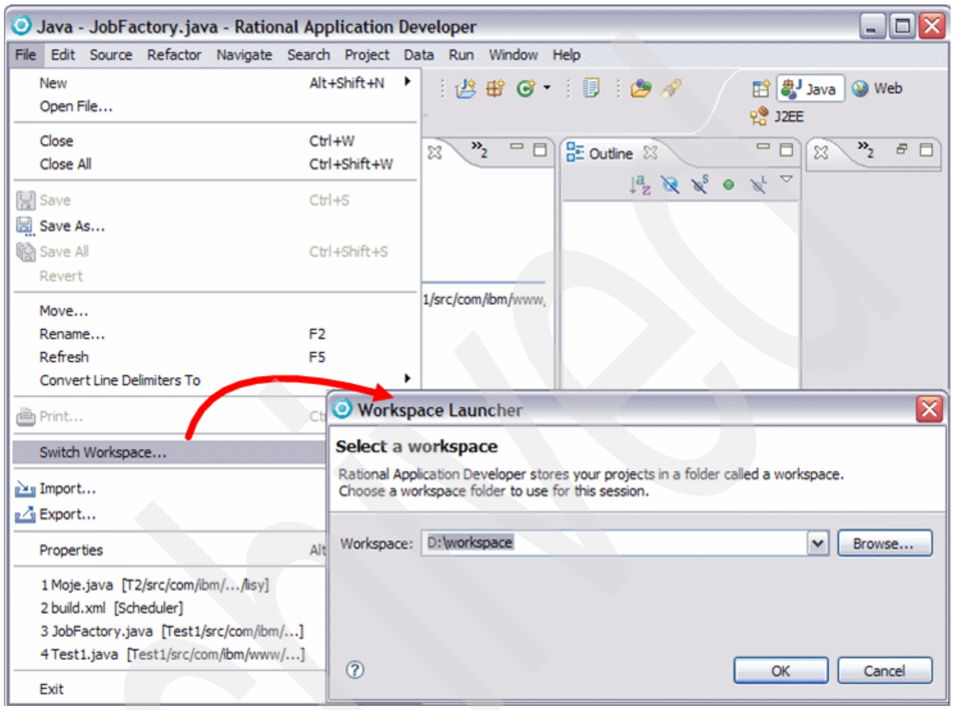
Figure 9-8 Adding a folder to the project - 2

4. Now you must copy the WSDL definitions together with other files to the wsdl subdirectory of our project. Perform the following copy:

   – Source: Select all of the WSDL files from the Tivoli Dynamic Workload Broker V1.2 installation CD, under the directory TDWB/wsdl.

   – Target: Copy the files into the <*workspace*>/Client/wsdl directory, where <*workspace*> is the directory on the file system that is dedicated for the workspace of your Rational Application Developer.

5. You can determine the workspace path by selecting **File** → **Switch workspace**. In the window that pops up, you can see the current workspace path (Figure 9-9). Do not change anything. Click **Cancel**. We just used one of possible ways of how to find out where we want to copy the WSDL files.



Select a workspace

Rational Application Developer stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\workspace

Figure 9-9 Determining the workspace path

6. When you have copied the files into the wsdl subdirectory of your project, return back to the Rational Application Developer workspace.

7. Now you must refresh the content of the wsdl directory within your project structure. In the Package Explorer pane right-click **wsdl** and select **Refresh** (Figure 9-10). You can refresh the view by pressing F5 as well.

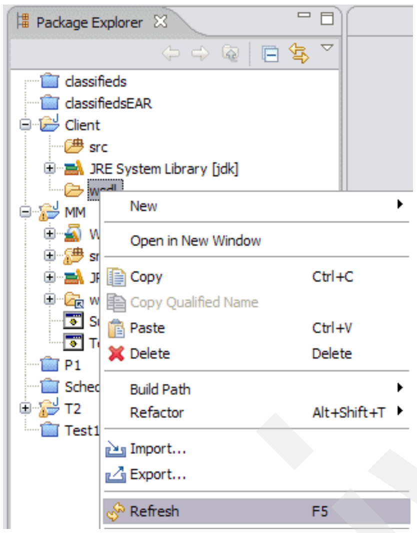


Figure 9-10 Refresh of wsdl folder

8. Expand the **wsdl** folder and see the result. The files copied from the installation media should be listed under the wsdl folder (Figure 9-11).

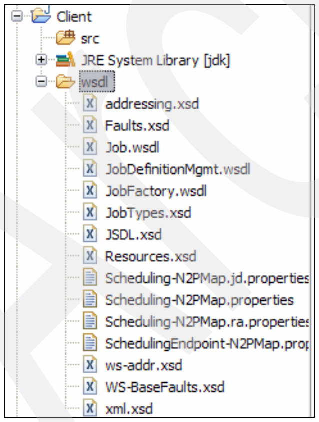


Figure 9-11 Copied files within the wsdl folder

Note: There is one more approach that can be used for providing the WSDL files to the project. You can just *link* the wsdl folder to the existing folder (during the creation of the new folder, you will click **Advanced** and then select **Link to folder in the file system**). Ponting to the directory by link is sufficient when you are sure that the files from the linked folder will always be available when needed. However, it is a good practice to keep the WSDL files (and related XML files) within the project structure. Keep in mind that if you linked the folder directly to teh CD media, the link will be broken when the CD is removed from the drive.

Creating the Web Service client

In this section we demonstrate how to create a Web Service client. We use the WSDL (and other important XML files) that we provided to the project in the previous step - “Providing the WSDL files to the project” on page 440.

1. In the Package Explorer pane right-click our **Client** project and select **New** → **Other** (Figure 9-12).

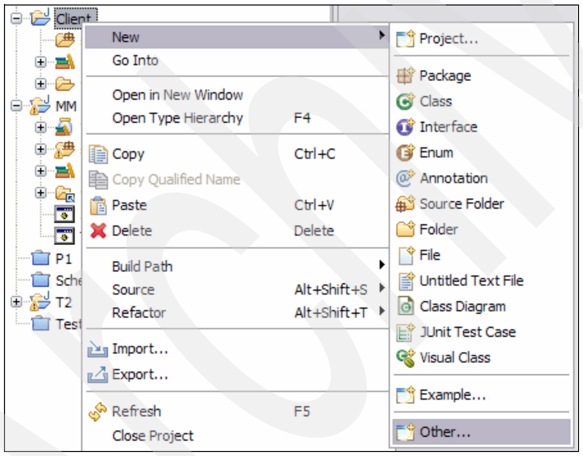
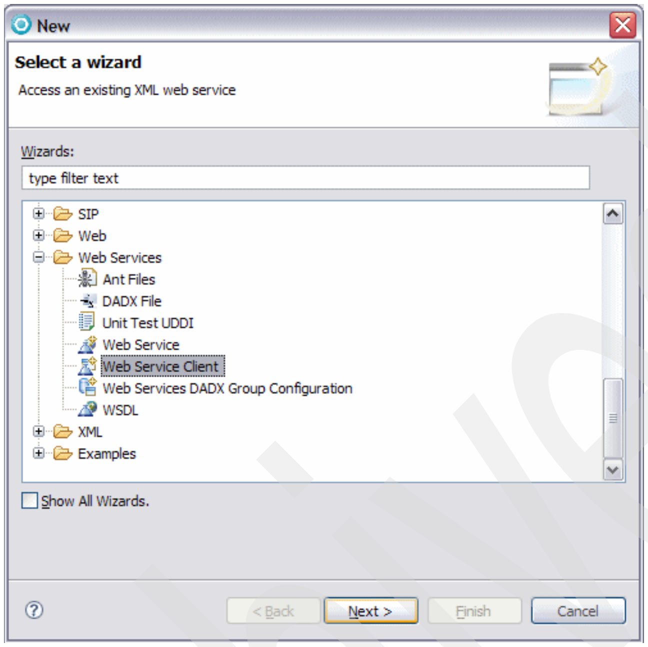


Figure 9-12 Creating the Web Service client - 1

2. Scroll down if necessary and select **Web Services** → **Web Service Client** (Figure 9-13).



New

Select a wizard

Access an existing XML web service

Wizards:

type filter text

- SIP
- Web
- Web Services
  - Ant Files
  - DADX File
  - Unit Test UDDI
  - Web Service
  - Web Service Client
  - Web Services DADX Group Configuration
  - WSDL
- XML
- Examples

Show All Wizards.

Figure 9-13 Creating the Web Service client - 2

3. Point to the correct WSDL file. Click **Browse** (Figure 9-14).

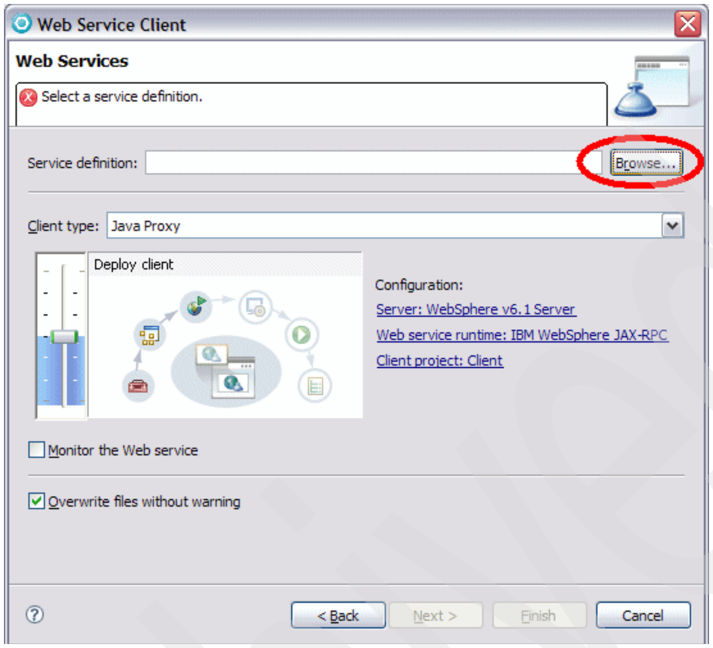


Figure 9-14 Creating the Web Service client - 3

4. In the following widow click **Browse** again. In the Window that pops up, navigate to **Client** → **wsdl** → **JobFactory.wsdl**. Click **OK** (Figure 9-15).

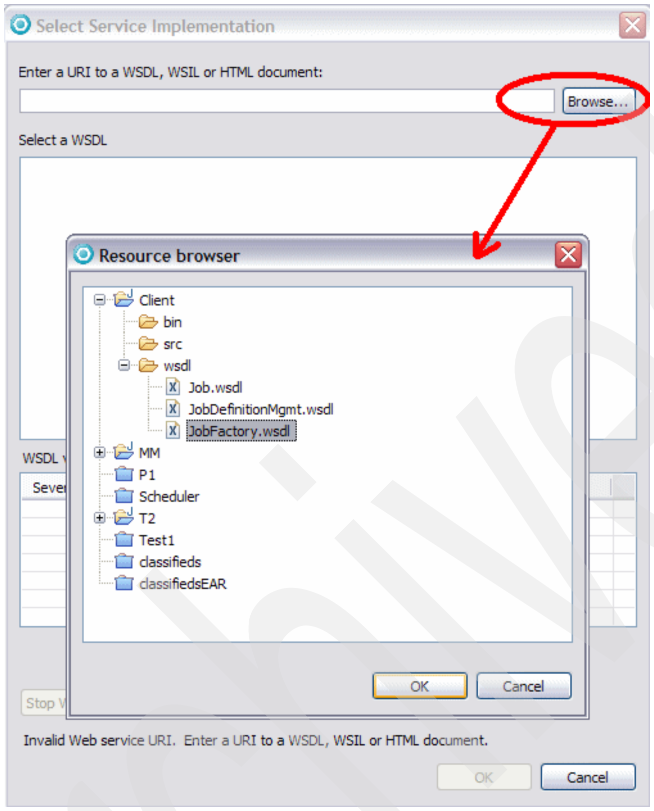


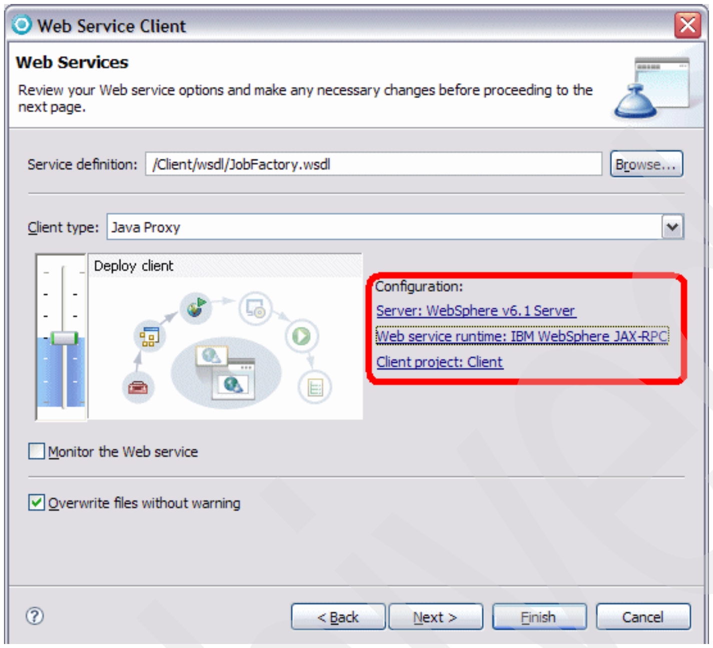
Figure 9-15 Creating the Web Service client - 4

5. Click **OK** and **OK** again.

   In the Web Service Client window check the configuration settings. They should be as follows:

   – Server: WebSphere V6.1 Server
   – Web service run time: IBM WebSphere JAX-RPC
   – Client project: Client

See Figure 9-16.



Web Service Client

Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Service definition: /Client/wsdl/JobFactory.wsdl

Client type: Java Proxy

Deploy client

Configuration:
Server: WebSphere v6.1 Server
Web service runtime: IBM WebSphere JAX-RPC
Client project: Client

☐ Monitor the Web service

☑ Overwrite files without warning

Figure 9-16   Creating the Web Service client-5

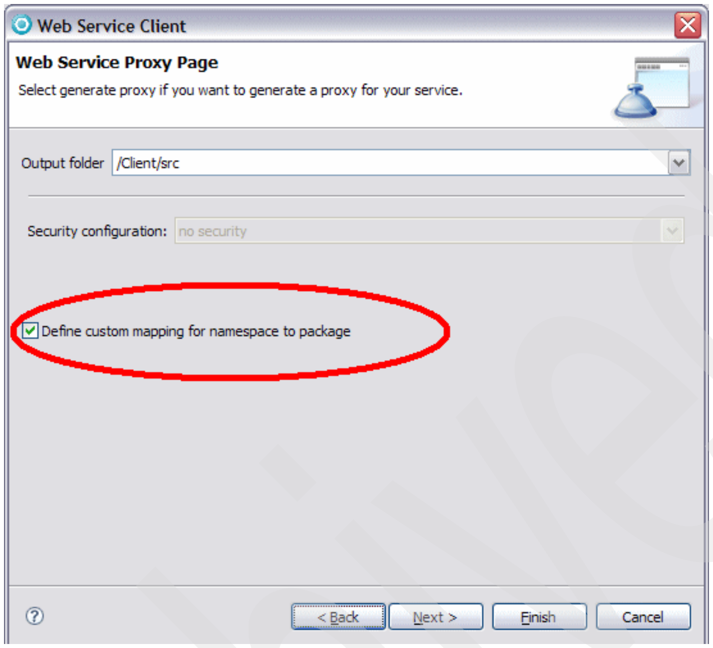
6. It the settings are different, adjust them so that they correspond to the settings shown above.

   Important: It is necessary to use WebSphere Application server V6.1. Prior versions are not supported for Tivoli Dynamic Workload Broker Web services client development.

7. Now remap the namespaces to different Java package naming. If we did not do that, the conversion tool would generate package names corresponding to namespaces defined within the JobFactory.wsdl file. The package names would be less readable and their nesting would not be that simple.

   There is a file provided with the WSDL files containing all the necessary mapping information. The name of the file is Scheduling-N2PMap.jd.properties. We use this file in our scenario in order to map namespaces to a corresponding package structure.

8. In the current window (Web Service Client), select **Define custom mapping for namespace to package** and click **Next** (Figure 9-17).



Select generate proxy if you want to generate a proxy for your service.

Output folder /Client/src

Security configuration: no security

[✓] Define custom mapping for namespace to package

Figure 9-17 Creating the Web Service client - 6

9. In the following window click **Import**. In the window that pops up navigate to **Client** → **wsdl** → **Scheduling-N2PMap.jd.properties** (Figure 9-18).

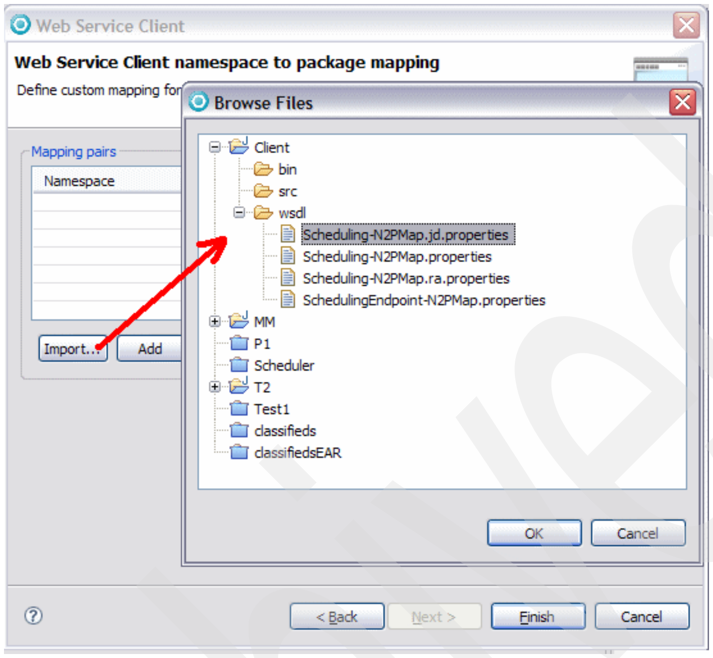
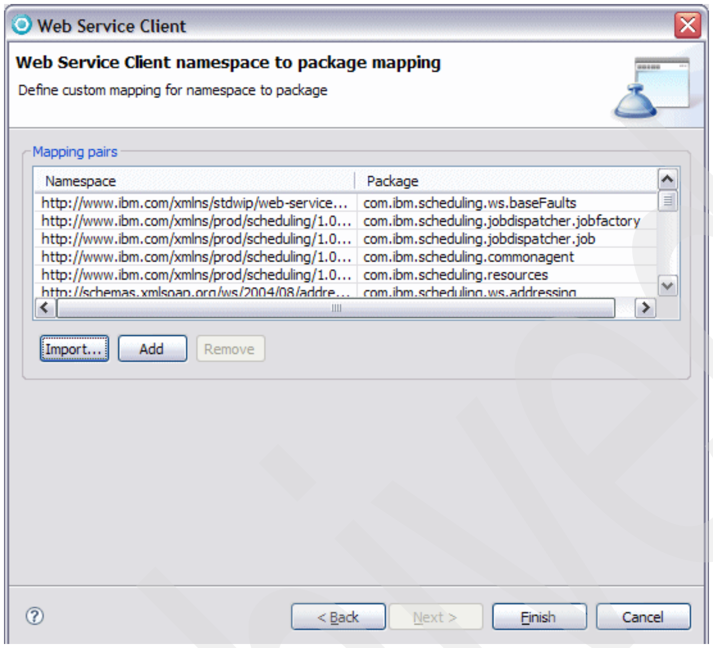

Web Service Client
Web Service Client namespace to package mapping
Define custom mapping for
Mapping pairs
Namespace
Import...
Add
Browse Files
Client
bin
src
wsdl
Scheduling-N2PMap.jd.properties
Scheduling-N2PMap.properties
Scheduling-N2PMap.ra.properties
SchedulingEndpoint-N2PMap.properties
MM
P1
Scheduler
T2
Test1
classifieds
classifiedsEAR
OK
Cancel
< Back
Next >
Finish
Cancel

Figure 9-18   Creating the Web Service client - 7

10. Click **OK**. You should see the mapping of namespaces into package names (Figure 9-19).

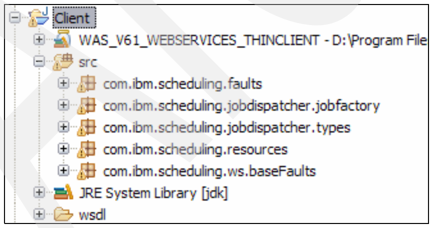

Web Service Client
Web Service Client namespace to package mapping
Define custom mapping for namespace to package
Mapping pairs
Import...
Add
Remove
< Back
Next >
Finish
Cancel

Namespace	Package
http://www.ibm.com/xmlns/stdwip/web-service...	com.ibm.scheduling.ws.baseFaults
http://www.ibm.com/xmlns/prod/scheduling/1.0...	com.ibm.scheduling.jobdispatcher.jobfactory
http://www.ibm.com/xmlns/prod/scheduling/1.0...	com.ibm.scheduling.jobdispatcher.job
http://www.ibm.com/xmlns/prod/scheduling/1.0...	com.ibm.scheduling.commonagent
http://www.ibm.com/xmlns/prod/scheduling/1.0...	com.ibm.scheduling.resources
http://schemas.xmlsoap.org/ws/2004/08/addre...	com.ibm.scheduling.ws.addressing

Figure 9-19 Creating the Web Service client-8

11. Click **Finish**. Click **OK** if you see the window with message `Warning messages were issued`.

12. In the workspace go to the Package Explorer pane. Expand the folder for sources (**src**) and look at the packages that were added automatically to your project.



```
Client
  WAS_V61_WEBSERVICES_THINCLIENT - D:\Program File
  src
    com.ibm.scheduling.faults
    com.ibm.scheduling.jobdispatcher.jobfactory
    com.ibm.scheduling.jobdispatcher.types
    com.ibm.scheduling.resources
    com.ibm.scheduling.ws.baseFaults
  JRE System Library [jdk]
  wsdl
```

Figure 9-20 Packages for Web services client

Now you have finished the necessary steps that generated Java packages, allowing you to use a JobFactory Web Service on the Tivoli Dynamic Workload Broker server from your client.

Adding the application logic

In this section we demonstrate how to create the package and the class that contain the main application logic. This *core application class* uses the Java packages generated in “Creating the Web Service client” on page 445.

We demonstrate the following tasks:

- Creating the package containing the core application class
- Importing the packages generated in previous step into the core application class
- Providing the source code using the Web Service client Java packages generated in the previous step

Note: We use the term *core application class* in this scenario. This is our terminology that helps us to distinguish the class that implements the application logic from the classes that were generated from the WSDL definitions. *Core application class* is not an official Java term.

Creating the package and Java class

In this section we describe how to create the package with a class that includes the application logic and references (imports) the Web Service client Java packages that we have generated from the WSDL file.

1. First we create a new package. In the Package Explorer navigate to the source folder under our project. Expand **Client** and right-click **src**. Select **New** → **Package** (Figure 9-21).

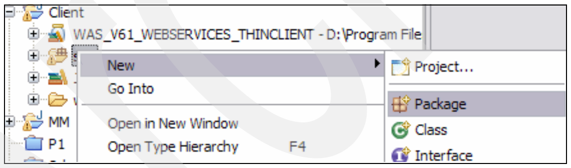


Figure 9-21 Creating a new package - 1

2. Type the package name. In this scenario we use com.ibm.scheduling.Submitter as the package name (Figure 9-22).

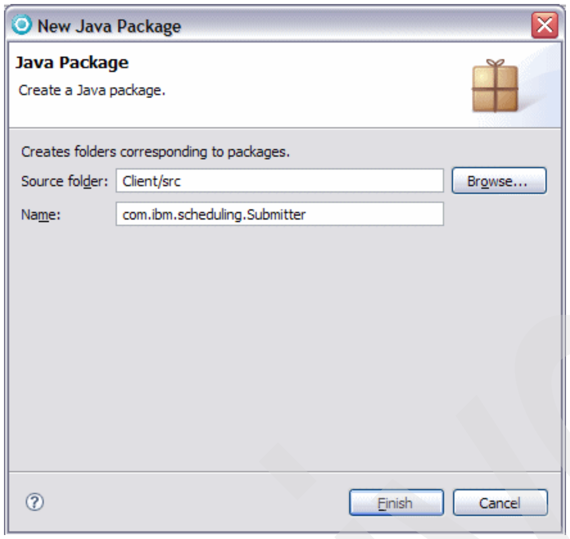

New Java Package
Java Package
Create a Java package.
Creates folders corresponding to packages.
Source folder: Client/src
Browse...
Name: com.ibm.scheduling.Submitter
Finish
Cancel

Figure 9-22 Creating a new package - 2

3. Click **Finish**. You should see the new package com.ibm.scheduling.Submitter in the Package Explorer. Right-click it and select **New** → **Class** (Figure 9-23).

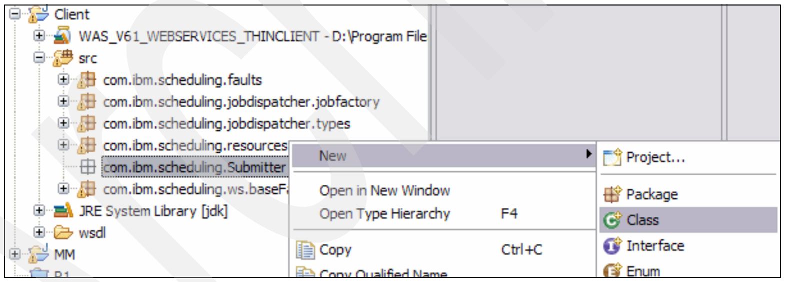
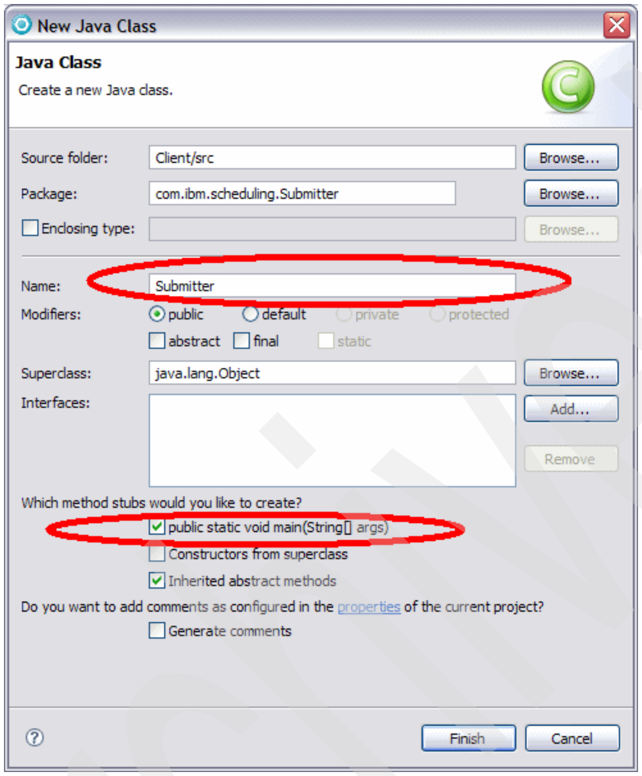

Client
WAS_V61_WEBSERVICES_THINCLIENT - D:\Program File
src
com.ibm.scheduling.faults
com.ibm.scheduling.jobdispatcher.jobfactory
com.ibm.scheduling.jobdispatcher.types
com.ibm.scheduling.resources
com.ibm.scheduling.Submitter
com.ibm.scheduling.ws.baseF
JRE System Library [jdk]
wsdl
MM
New
Open in New Window
Open Type Hierarchy F4
Copy Ctrl+C
Copy Qualified Name
Project...
Package
Class
Interface
Enum

Figure 9-23 Creating a new class - 1

4. Type the class name. In our scenario we use Submitter as the class name. Select the check box **public static void main(String[] args)** to include the template of the main method in the class definition (Figure 9-24).



New Java Class

Java Class

Create a new Java class.

Source folder: Client/src

Package: com.ibm.scheduling.Submitter

[ ] Enclosing type:

Name: Submitter

Modifiers: [● public] ○ default ○ private ○ protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

[ ] Constructors from superclass

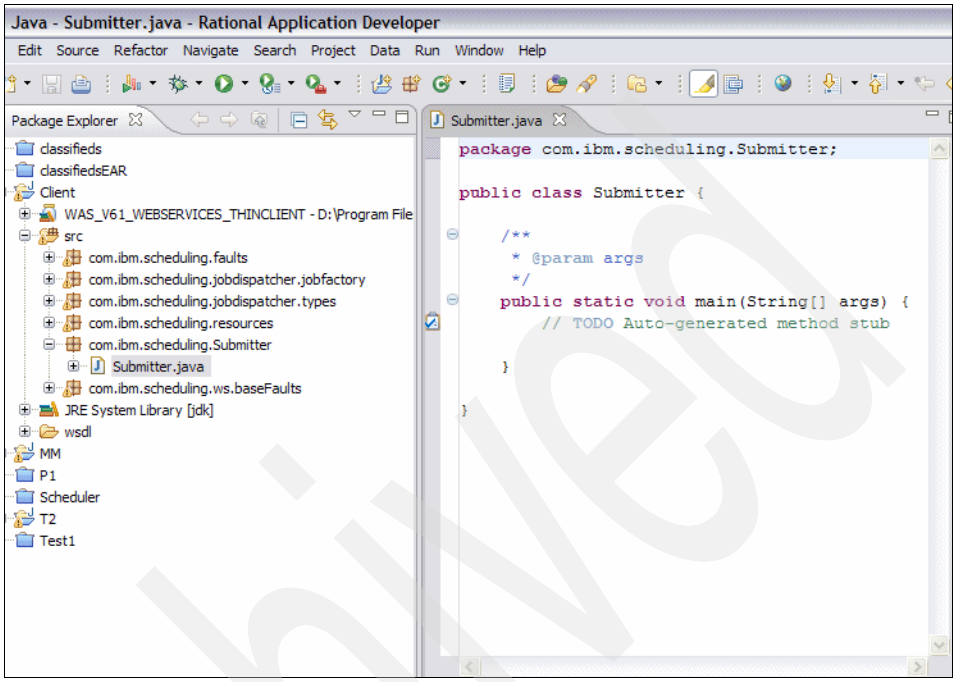
[✓] Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

[ ] Generate comments

Figure 9-24 Creating a new class - 2

5. Click **Finish**. The generated class should appear in your workspace. Figure 9-25 shows the workspace with the generated class.



Java - Submitter.java - Rational Application Developer
Edit Source Refactor Navigate Search Project Data Run Window Help
Package Explorer
classifieds
classifiedsEAR
Client
WAS_V61_WEBSERVICES_THINCLIENT - D:\Program File
src
com.ibm.scheduling.faults
com.ibm.scheduling.jobdispatcher.jobfactory
com.ibm.scheduling.jobdispatcher.types
com.ibm.scheduling.resources
com.ibm.scheduling.Submitter
Submitter.java
com.ibm.scheduling.ws.baseFaults
JRE System Library [jdk]
wsdl
MM
P1
Scheduler
T2
Test1
Submitter.java

```
package com.ibm.scheduling.Submitter;

public class Submitter {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Figure 9-25 Workspace with the generated class

Importing the generated packages

In this section we describe which generated Web Service client Java packages should be imported into the core application class. We list the packages that must be imported in order to use the JobFactory Web service.

According to the namespace-to-package mapping that we specified by using the Scheduling-N2PMap.jd.properties file (while generating the packages from WSDL definitions), the package names begin with the com.ibm.scheduling.jobdispatcher prefix.

To allow the sample client to use classes defined within these packages, you must *import* the packages into the core application class that will reference them. Importing allows you to reference to the classes without referencing the package name. Example 9-32 shows which statements are used for importing the packages generated from the JobFactory.wsdl file.

Example 9-32 Importing the packages

```
import com.ibm.scheduling.jobdispatcher.jobfactory.JobFactory;
import
com.ibm.scheduling.jobdispatcher.jobfactory.JobFactoryServiceLocator;
import com.ibm.scheduling.jobdispatcher.jobfactory.JobStatus;
import com.ibm.scheduling.jobdispatcher.types.JobStateEnumeration;
```

The Client application source code

In this section we provide the entire source code of the client core application class. This is the Submitter class defined within the com.ibm.scheduling.Submitter package.

The entire code is included in Example 9-33. The code has the important parts fully commented.

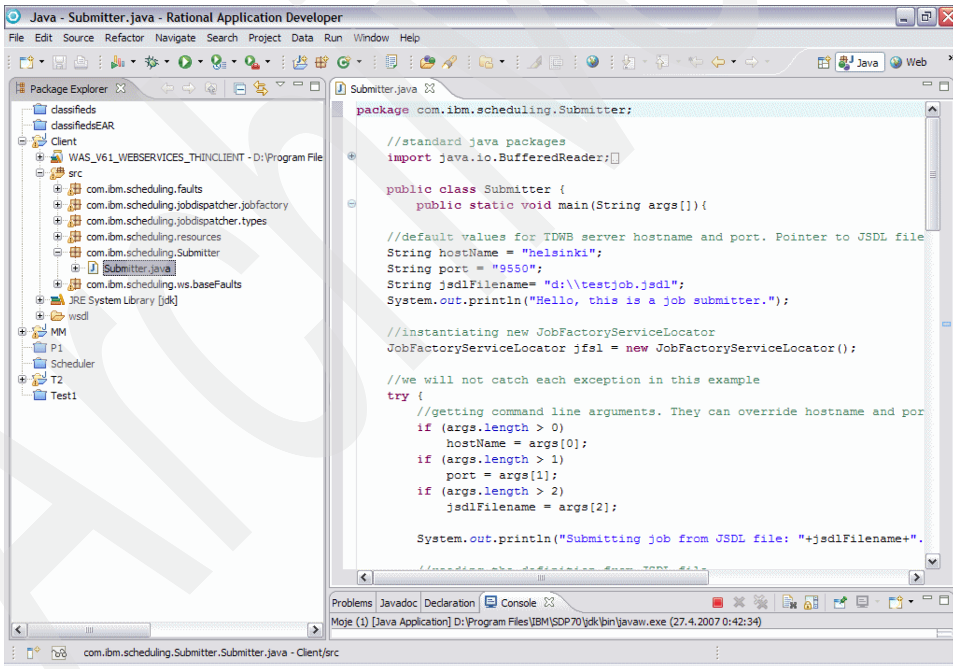
Example 9-33 Core application class with main method

```
package com.ibm.scheduling.Submitter;

   //standard java packages
   import java.io.BufferedReader;
   import java.io.FileInputStream;
   import java.io.InputStream;
   import java.io.InputStreamReader;

   //our packages generated by WSDL2Java from WSDL definition
   //default namespaces were remapped to packages in hierarchy com.inm.scheduling
   import com.ibm.scheduling.jobdispatcher.jobfactory.JobFactory;
   import com.ibm.scheduling.jobdispatcher.jobfactory.JobFactoryServiceLocator;
   import com.ibm.scheduling.jobdispatcher.jobfactory.JobStatus;
   import com.ibm.scheduling.jobdispatcher.types.JobStateEnumeration;

   public class Submitter {
      public static void main(String args[]){

    //default values for TDWB server hostname and port. Pointer to JSDL file on local
filesystem.
   String hostName = "helsinki";
   String port = "9550";
   String jsdlFilename= "d:\\testjob.jsdl";
   System.out.println("Hello, this is a job submitter.");
```

```
    //instantiating new JobFactoryServiceLocator
    JobFactoryServiceLocator jfsl = new JobFactoryServiceLocator();

    //we will not catch each exception in this example
    try {
        //getting command line arguments. They can override hostname and port of TDWB server
and JSDL file name
       if (args.length > 0)
          hostName = args[0];
       if (args.length > 1)
          port = args[1];
       if (args.length > 2)
          jsdlFilename = args[2];

       System.out.println("Submitting job from JSDL file: "+jsdlFilename+". TDWB hostname:
"+hostName+"TDWB port"+port);

        //reading the definition from JSDL file
        StringBuffer document = new StringBuffer();
        InputStream is = null;
        System.out.println("Reading jsdl file " + jsdlFilename);
        is = new FileInputStream(jsdlFilename);
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);
        String line = null;
        while ((line = br.readLine()) != null) {
            document.append(line);
        } //while

        //creating URLs for JobFactory
       String jobFactoryURL = "http://" + hostName + ":" + port +
"/JDServiceWS/services/JobFactory";

        //instantiating new jobFactory with corresponding URL
        JobFactory jf = jfsl.getJobFactory(new java.net.URL(jobFactoryURL));

       System.out.println("Submitting job");

       // submitting the job, using the JobFactory method submitJobFromJSDLXml()
       // getting the endpoint reference of that job to "epr". We will use "epr" later on,
       // whenever we will need to point to the instance of submitted job
       // operation submitJobFromJSDLXml requires several parameters, the only necessary for
our purpose
       // is the JobDefinitionDocument (described in the type definition of
submitJobFromJSDLXml operation)
       // the other 5 arguments are referrenced as null
       com.ibm.websphere.wsaddressing.EndpointReference epr =
jf.submitJobFromJSDLXml(document.toString(),
               null,null,null,null,null);

       //printing EndpointReference to the submitted job
       System.out.println("Succesfull epr: " + epr);

       //instantiating an array of endpoint references
       com.ibm.websphere.wsaddressing.EndpointReference jobEprs[] = new
com.ibm.websphere.wsaddressing.EndpointReference[1];

       //store the EndpointReference to the 1st position of the array
       //we need the array of Endpoint references for querying job states,
       //because the method getJobsStatuses() (a few lines below)
       //requires an array of EndpointReferences as its argument
       jobEprs[0] = epr;

       //beginning of loop
       boolean term = true;
       do {
          //querying the job status.
          //pass the ARRAY of endpoint references to the JobFactory method getJobsStatuses()
          //method getJobsStatuses() returns an array of job states as a response.
          //store the response into array "gjsResp" of type JobStatus
          JobStatus[] gjsResp = jf.getJobsStatuses(jobEprs);

          //if we have at least one job status in the array of job states
          if (gjsResp.length > 0) {
             //call the method getState() of JobStatus (instantiated to "gjsResp")
             //query the 1st position of the array. It corresponds to the 1st position
             //of our endpoint reference
             System.out.println("Job status: " + gjsResp[0].getState());
             //test, if there was not any fault while getting the job state
             if (gjsResp[0].getFault() == null) {
                //now compare the status of the jobs against the list of desired states
                //until the job that we have submitted, does not reach the desired state, the
loop will not end.
                term = (gjsResp[0].getState() == JobStateEnumeration.SUCCEEDED_EXECUTION) ||
                (gjsResp[0].getState() == JobStateEnumeration.FAILED_EXECUTION) ||
                (gjsResp[0].getState() == JobStateEnumeration.NOT_EXECUTED) ||
                (gjsResp[0].getState() == JobStateEnumeration.RESOURCE_ALLOCATION_FAILED);
                Thread.sleep(1000);
             } else {
                //if fault occured, print error code
                System.out.println("Error: " + gjsResp[0].getFault().getErrorCode());
                term = true;
             } //else (on getfault)
          } else {
             System.out.println("Error empty vector");
             term = true;
          } //else (on gjsResp.length)
       } while (!term);  //end of loop

       System.out.println("Succesfull");

       } catch (Exception e) {
          //we do not determine the exact exception...
       e.printStackTrace();
       } //catch
    } //main
} //Submitter
```

Paste this code into the Submitter.java class. Replace all of the previously generated content of the Submitter.java class, because the code provided in this example already contains all of the necessary content.

After putting this code into Submitter.java, your workspace should look similar to Figure 9-26. We minimized some unimportant workspace windows because we want to show you the most important information in this figure.


Java - Submitter.java - Rational Application Developer
File Edit Source Refactor Navigate Search Project Data Run Window Help
Java
Web
Package Explorer
classifieds
classifiedsEAR
Client
WAS_V61_WEBSERVICES_THINCLIENT - D:\Program File
src
com.ibm.scheduling.faults
com.ibm.scheduling.jobdispatcher.jobfactory
com.ibm.scheduling.jobdispatcher.types
com.ibm.scheduling.resources
com.ibm.scheduling.Submitter
Submitter.java
com.ibm.scheduling.ws.baseFaults
JRE System Library [jdk]
wsdl
MM
P1
Scheduler
T2
Test1
Submitter.java
Problems Javadoc Declaration Console
com.ibm.scheduling.Submitter.Submitter.java - Client/src

```
package com.ibm.scheduling.Submitter;

    //standard java packages
    import java.io.BufferedReader;

    public class Submitter {
        public static void main(String args[]){

    //default values for TDWB server hostname and port. Pointer to JSDL file
    String hostName = "helsinki";
    String port = "9550";
    String jsdlFilename= "d:\\testjob.jsdl";
    System.out.println("Hello, this is a job submitter.");

    //instantiating new JobFactoryServiceLocator
    JobFactoryServiceLocator jfsl = new JobFactoryServiceLocator();

    //we will not catch each exception in this example
    try {
        //getting command line arguments. They can override hostname and por
        if (args.length > 0)
            hostName = args[0];
        if (args.length > 1)
            port = args[1];
        if (args.length > 2)
            jsdlFilename = args[2];

        System.out.println("Submitting job from JSDL file: "+jsdlFilename+".
```

```
Moje (1) [Java Application] D:\Program Files\IBM\SDP70\jdk\bin\javaw.exe (27.4.2007 0:42:34)
```

Figure 9-26   Putting the code into Submitter.java

Providing the sample JSDL file

In order to be able to test the functionality of the sample client, you must point to the existing JSDL file located somewhere in your file system. You can create your own sample definition or use any existing JSDL file.

In Example 9-34 we provide a sample JSDL definition that you can use for testing purposes. In our scenario we have this definition stored in the D:\testjob.jsdl file.

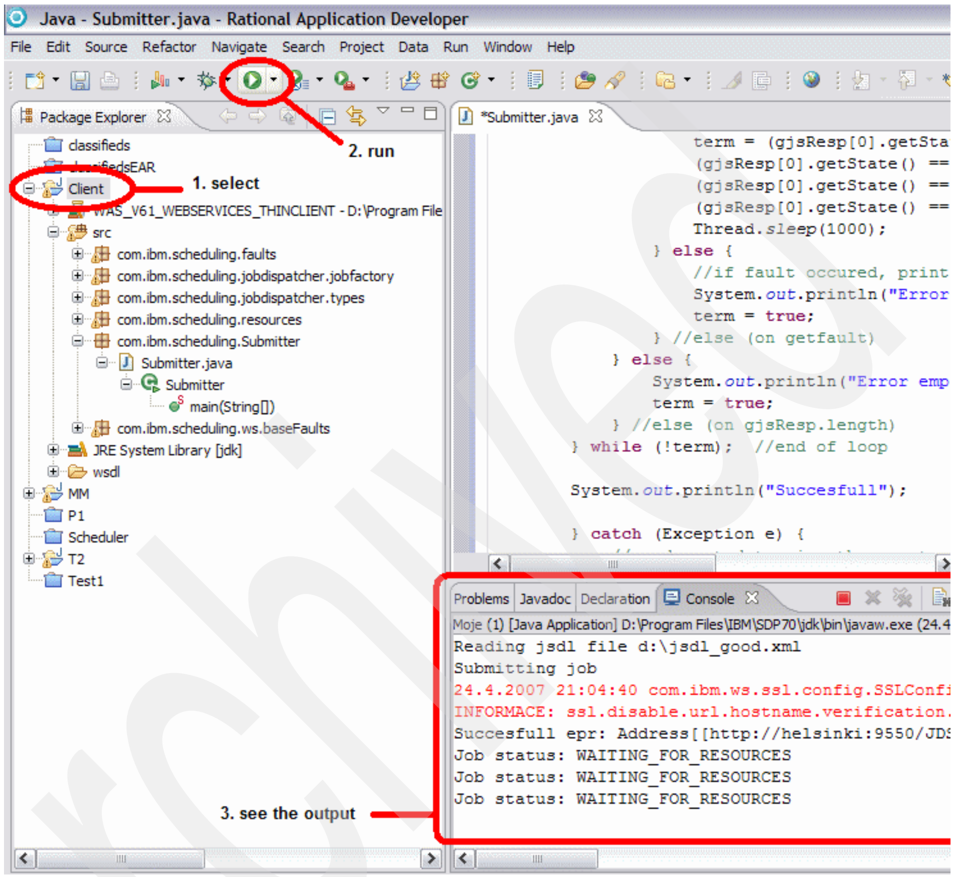
Example 9-34 Sample JSDL definition

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl JSDL.xsd
http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle JSDL-Native.xsd" description="Sample Test
job" name="testjob">
  <jsdl:annotation>This job is for testing purposes only. It does not require any special
resource to run.</jsdl:annotation>
  <jsdl:application name="executable">
    <jsdle:executable path="echo TDWB_SAMPLE_JOB"/>
  </jsdl:application>
</jsdl:jobDefinition>
```

Running the sample client from the Rational Application Developer

In this section we describe how to run the sample client from the Rational Application Developer environment.

In the Package Explorer pane navigate to our project named Client. Then click the Run icon, as shown in Figure 9-27. In the console pane you can see the output of the project. You should see similar output as that shown in Figure 9-27.


Java - Submitter.java - Rational Application Developer
File Edit Source Refactor Navigate Search Project Data Run Window Help
Package Explorer
*Submitter.java
classifieds
ClassifiedsEAR
Client
WAS_V61_WEBSERVICES_THINCLIENT - D:\Program File
src
com.ibm.scheduling.faults
com.ibm.scheduling.jobdispatcher.jobfactory
com.ibm.scheduling.jobdispatcher.types
com.ibm.scheduling.resources
com.ibm.scheduling.Submitter
Submitter.java
Submitter
main(String[])
com.ibm.scheduling.ws.baseFaults
JRE System Library [jdk]
wsdl
MM
P1
Scheduler
T2
Test1
1. select
2. run
3. see the output
Problems Javadoc Declaration Console

```
                term = (gjsResp[0].getSta
                (gjsResp[0].getState() ==
                (gjsResp[0].getState() ==
                (gjsResp[0].getState() ==
                Thread.sleep(1000);
            } else {
                //if fault occured, print
                System.out.println("Error
                term = true;
            } //else (on getfault)
        } else {
            System.out.println("Error emp
            term = true;
        } //else (on gjsResp.length)
    } while (!term);  //end of loop

    System.out.println("Succesfull");

    } catch (Exception e) {
```

```
Moje (1) [Java Application] D:\Program Files\IBM\SDP70\jdk\bin\javaw.exe (24.4
Reading jsdl file d:\jsdl_good.xml
Submitting job
24.4.2007 21:04:40 com.ibm.ws.ssl.config.SSLConfi
INFORMACE: ssl.disable.url.hostname.verification.
Succesfull epr: Address[http://helsinki:9550/JDS
Job status: WAITING_FOR_RESOURCES
Job status: WAITING_FOR_RESOURCES
Job status: WAITING_FOR_RESOURCES
```

Figure 9-27 Running the sample client within Rational Application Developer

For the instructions on how to run the sample client from the command line see 9.5.4, “Running the sample client from the command line” on page 485.

9.5.3 Creating the sample client using Eclipse

In this section we describe how to create a sample client using the Eclipse V3.2.1 together with utilities and JAR files provided with WebSphere Application Server V6.1.

We describe the actions that must be taken in order to successfully create the sample client from scratch. The step-by-step instructions describe following tasks:

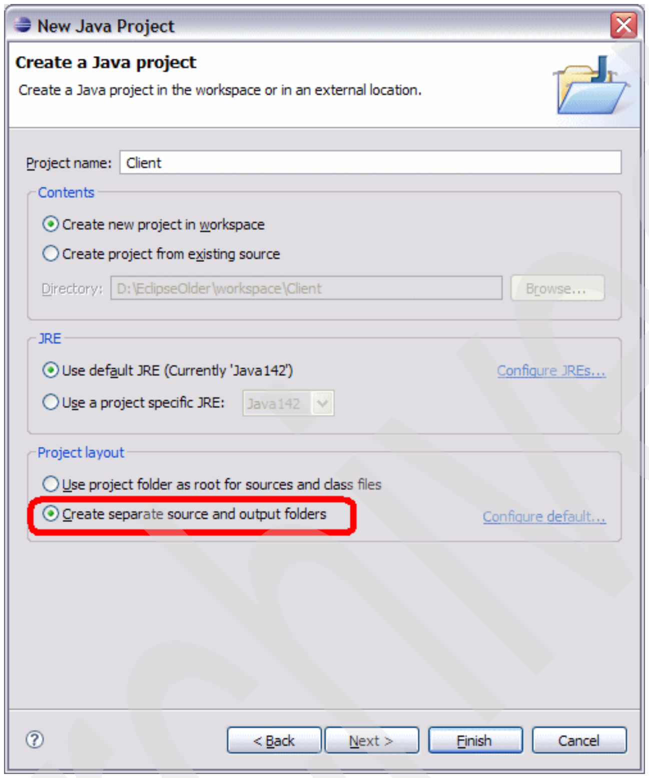
- Creating a project
- Creating the Web Service client (generating Java packages from WSDL definition using the WSDL2Java command-line utility)
- Providing the Java packages created in the previous step to the project
- Adding the application logic and linking it together with the Java packages created in the previous step
- Running the project

Creating the project

In this section we demonstrate how to create the project for our sample client.

1. Launch Eclipse and create a new project by selecting**File** → **New Project**. Select **Java project** and click **Next**.

2. In the following window type in the project name. Type `Client` into the Project Name text box. Then specify separate subdirectories for the source code and for the output (Java classes). Select **Create separate source and output folders** (Figure 9-28).



New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: Client

Contents

- (●) Create new project in workspace
- ( ) Create project from existing source

Directory: D:\EclipseOlder\workspace\Client

JRE

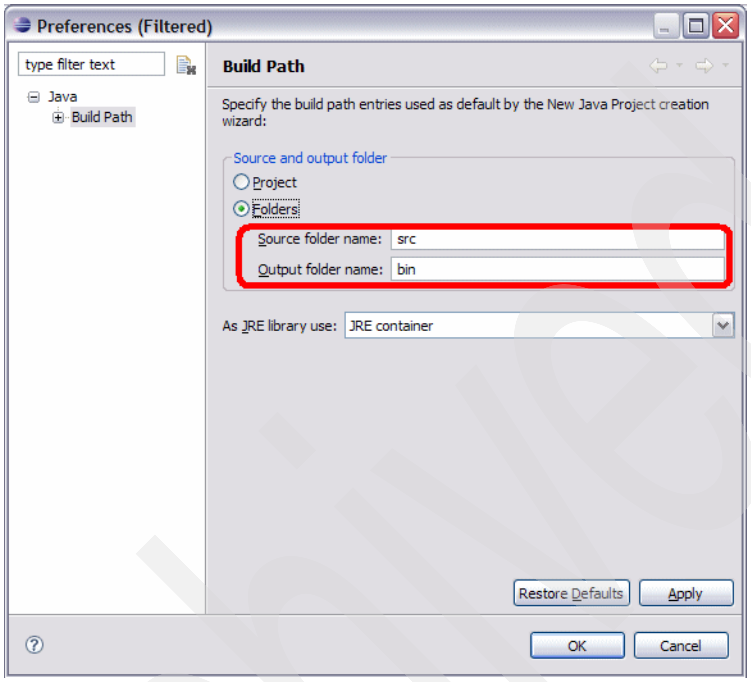
- (●) Use default JRE (Currently 'Java142')
- ( ) Use a project specific JRE: Java142

Project layout

- ( ) Use project folder as root for sources and class files
- (●) Create separate source and output folders

Figure 9-28 Creating a new Java Project

3. To make sure which directories are dedicated for the input (sources) and output (classes), click **Configure default**, and in the following window check the values of source folder name and output folder name (Figure 9-29).


Preferences (Filtered)

type filter text

Java
Build Path

Build Path

Specify the build path entries used as default by the New Java Project creation wizard:

Source and output folder
- Project
- [✓ Folders]

Source folder name: src
Output folder name: bin

As JRE library use: JRE container

Figure 9-29 Selecting the source and output folders

4. Keep the following values:
   – Source folder name: src
   – Output folder name: bin
5. Click **OK** to close the Preferences window and click **Next** in the following window.

   Now you must provide the project with the additional JAR file containing the necessary Java classes necessary for building and running the Web services client. This JAR file is included in the WebSphere Application Server V6.1 installation directory. You must add the <*was_install_dir*>/runtimes/com.ibm.ws.webservices.thinclient_6.1.0.jar to the project.

The default WebSphere Application Server installation path is as follows:

- Windows - C:\Program Files\IBM\WebSphere\AppServer
- AIX - /usr/IBM/WebSphere/AppServer
- Other UNIX platforms and Linux - /opt/IBM/WebSphere/AppServer

To provide the project with additional JARs do the steps described below.

6. While still in the New Java Project window, click the **Libraries** folder and then click **Add External JARs** (Figure 9-30).

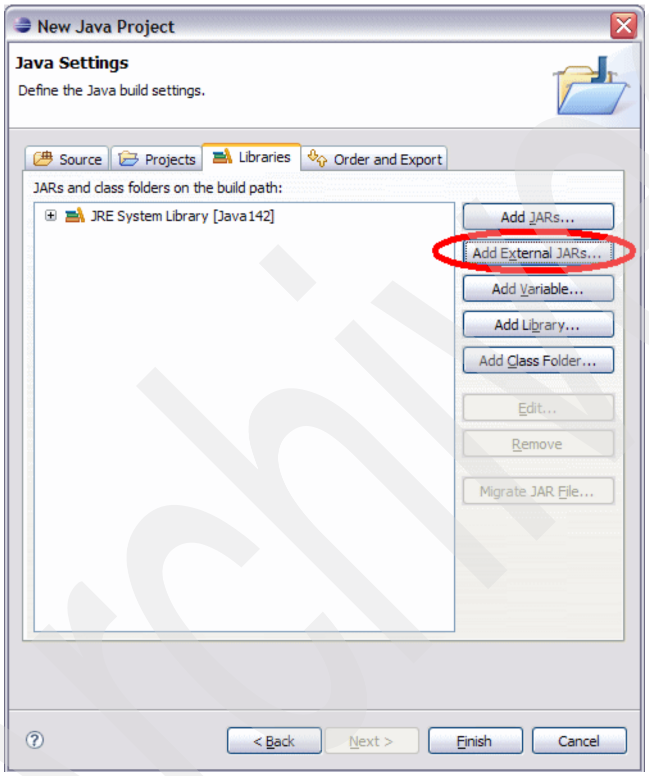
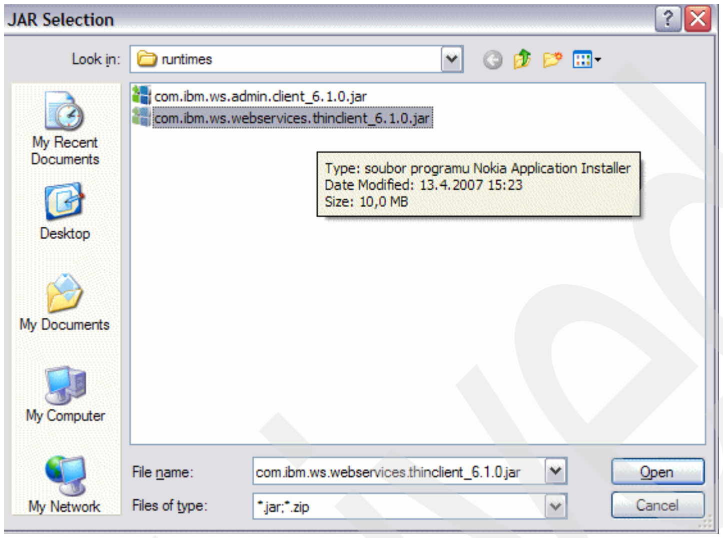


Figure 9-30 Adding JAR to the project - 1

7. A JAR selection browser pops up. Navigate to the *<was_install_dir>*/runtimes/ directory and select file **com.ibm.ws.webservices.thinclient_6.1.0.jar**. Click **Open** (Figure 9-31).


JAR Selection
Look in: runtimes
com.ibm.ws.admin.client_6.1.0.jar
com.ibm.ws.webservices.thinclient_6.1.0.jar
My Recent Documents
Desktop
My Documents
My Computer
My Network
File name: com.ibm.ws.webservices.thinclient_6.1.0.jar
Files of type: *.jar;*.zip
Open
Cancel

Type: soubor programu Nokia Application Installer
Date Modified: 13.4.2007 15:23
Size: 10,0 MB

Figure 9-31 Adding JAR to the project-2

8. Click **Finish** in the New Java Project window.

9. Based on your configuration, you should have your workspace divided into several panes. If you do not see the Package Explorer pane in your workspace, switch the view by selecting **Window** → **Show view** → **Package Explorer** (Figure 9-32).

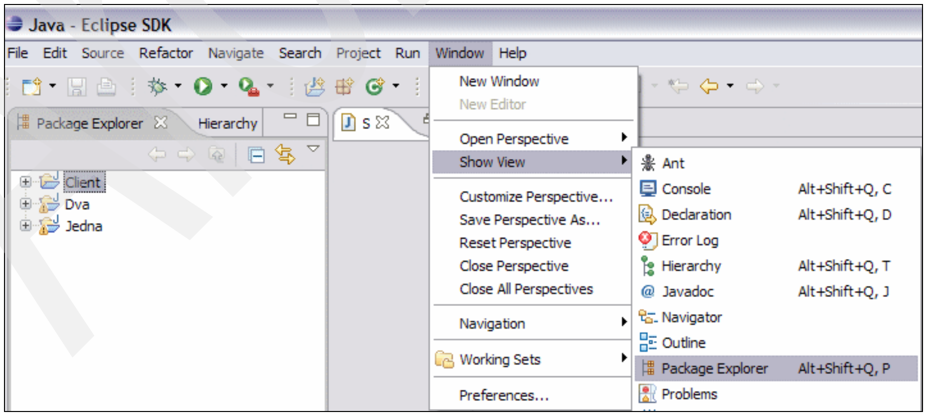


Figure 9-32 Switching to Package Explorer view

Now your workspace should look like Figure 9-33.

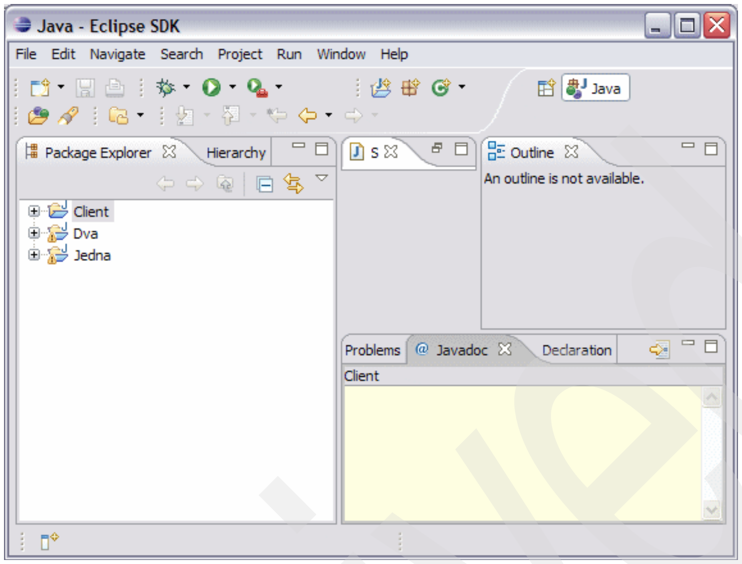


Figure 9-33 Eclipse - workspace

Generating the Web Service java packages from the WSDL definition

In this section we describe how to build the Web services client Java packages from the WSDL files using the command-line utility WSDL2Java.

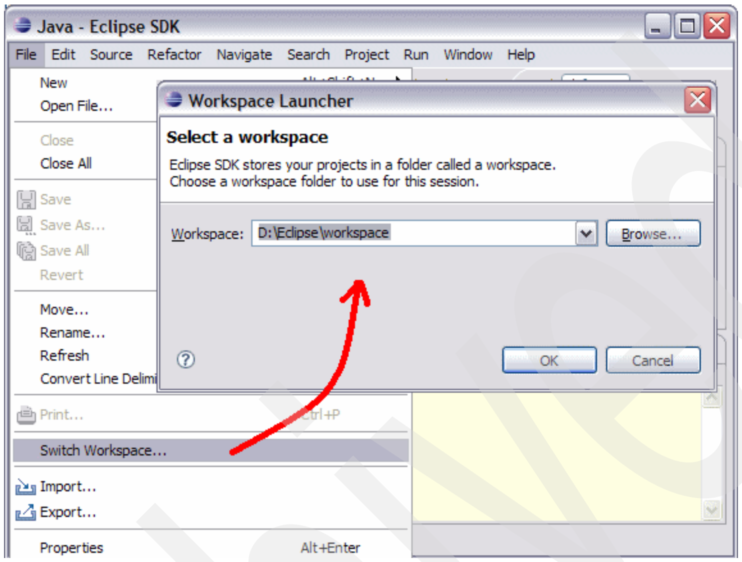
1. Prior to running the tool you should know the location of the WSDL files. We recommend that you copy the files from the installation media to your hard drive.

   Perform the following copy:

   – Source: Select all of the WSDL files from the Tivoli Dynamic Workload Broker V1.2 installation CD, under the directory TDWB/wsdl.

   – Target: Copy the files into the <*workspace*>/Client/wsdl directory, where <*workspace*> is the directory on the file system that is dedicated for the Eclipse workspace.

2. You can determine the workspace path by selecting **File** → **Switch workspace**. In the window that pops up, you can see the current workspace path (Figure 9-34). Do not change anything. Click **Cancel**. We just used one of possible ways of finding out where we want to copy the WSDL files.



Select a workspace

Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Eclipse\workspace

Figure 9-34 Eclipse - Determining the workspace path

WSDL2Java must be supplied with some necessary parameters in order to generate the proper packages.

An important optional parameter allows the remapping of the namespaces used in the WSDL definition to different Java package naming. If we did not use this parameter, the WSDL2Java conversion tool would generate package names corresponding to namespaces defined within the JobFactory.wsdl file. The package names would be less readable and their nesting would not be that simple.

There is a file provided with the WSDL files containing all of the necessary mapping information. The name of the file is Scheduling-N2PMap.jd.properties. We use this file in our scenario in order to map namespaces to a corresponding package structure.

3. Example 9-35 shows the sample syntax of the WSDL2Java utility, which is provided with the WebSphere Application Server V6.1. In order to generate the Web Service client for our scenario, you must supply the following parameters to the WSDL2Java utility:

   – -c client
   – -r client
   – - fileNStoPkg *<wsdl_files_directory>*/Scheduling-N2PMap.jd.properties
   – -o *<output_directory>*
   – *<wsdl_files_directory>/proper_wsdl_file*

   Note: If you have copied the files into the /wsdl subdirectory in your Client project, the value of *<wsdl_files_directory>* is *<workspace>*/Client/wsdl, where *<workspace>* is the location of your Eclipse workspace.

   In our scenario we use *<workspace>*/Client/services as the value for *<output_directory>.* This is not mandatory, but we recommend that you place the generated Web services client Java packages in to the subdirectory of your project file structure.

4. Example 9-35 demonstrates how the Web services client Java packages can be generated. If you receive similar warning messages to those shown in the example, you may ignore them.

Example 9-35 Generating the Java packages from the command line

```
C:\Program Files\IBM\WebSphere\AppServer\bin>WSDL2Java.bat -c client -r client -
fileNStoPkg D:\Eclipse\workspace\Client\wsdl\Scheduling-N2PMap.jd.properties -o
D:\Eclipse\workspace\Client\services D:\Eclipse\workspace\Client\wsdl\JobFactory
.wsdl

WSWS3752I: (C) COPYRIGHT International Business Machines Corp. 1997, 2006.
WSWS3753I: IBM WebSphere Application Server Release 6.1
WSWS3755I: Web services WSDL2Java emitter.

WSWS3029W: Warning: The xml construct named {http://www.ibm.com/xmlns/prod/sched
uling/1.0/job-management/job-factory}AffinityType cannot be mapped to a java typ
e.  The construct will be mapped to javax.xml.soap.SOAPElement.
WSWS3029W: Warning: The xml construct named {http://www.ibm.com/xmlns/prod/sched
uling/1.0/jsdl}ExtensibleElementsType cannot be mapped to a java type.  The cons
truct will be mapped to javax.xml.soap.SOAPElement.
WSWS3029W: Warning: The xml construct named {http://www.ibm.com/xmlns/prod/sched
uling/1.0/jsdl}VariablesType cannot be mapped to a java type.  The construct wil
l be mapped to javax.xml.soap.SOAPElement.
```

Note: The WSDL2Java utility is located in the *<was_install_dir>*/bin directory.

Now you have finished the necessary steps that generated the Web Service client Java packages allowing you to use a JobFactory Web Service on the Tivoli Dynamic Workload Broker server from your client.

Providing the Web Service client Java packages to the project

In this section we demonstrate how to provide the generated Web Service client Java package to the project.

1. In the Package Explorer pane navigate to our **Client** project. Right-click it and select **Refresh** (Figure 9-35). You can refresh the view by pressing F5 as well.

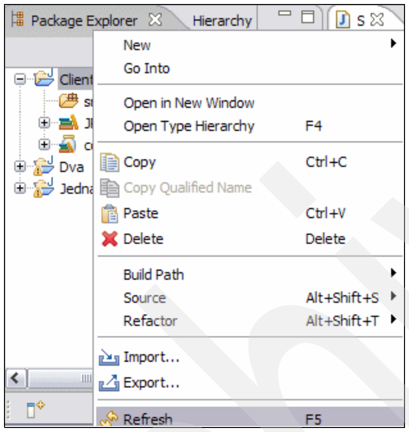
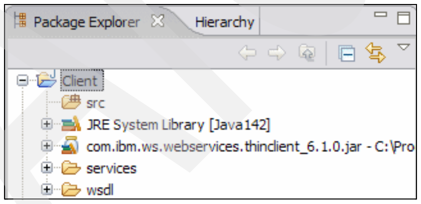


Figure 9-35 Refresh of the project content

2. If you expand the Client project, you should see that two directories have been added — one containing the original WSDL files and the second containing the generated Web Service Java packages (Figure 9-36).



Package Explorer
Hierarchy
Client
src
JRE System Library [Java142]
com.ibm.ws.webservices.thinclient_6.1.0.jar - C:\Pro
services
wsdl

Figure 9-36 Refreshed project content

3. In the Package Explorer pane navigate to our **Client** project. Right-click it and select **Properties** (Figure 9-37).

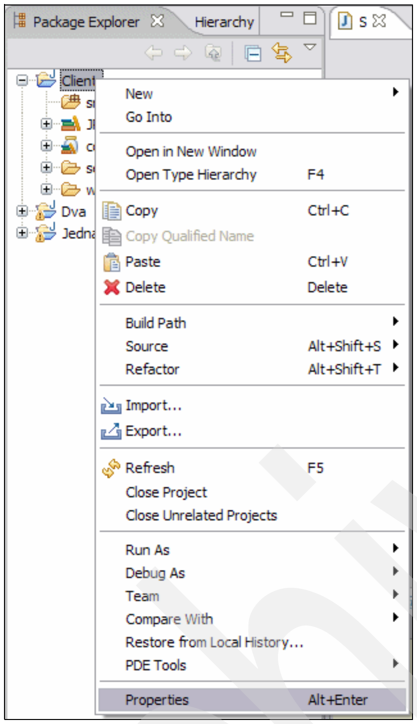
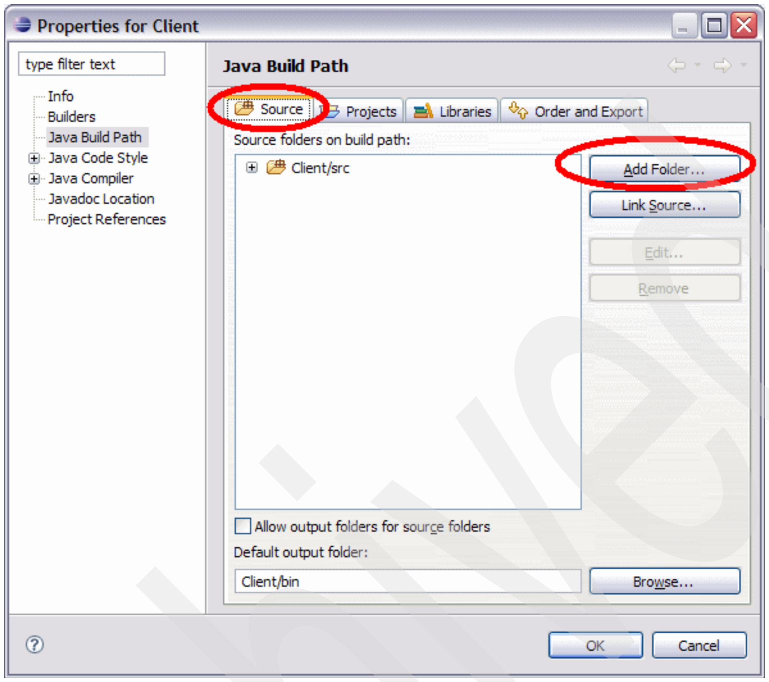


Figure 9-37 Opening the project properties

4. In the window that pops up, navigate to the Source folder. Click **Add Folder** (Figure 9-38).



Source folders on build path:

Client/src

Allow output folders for source folders

Default output folder:

Client/bin

Figure 9-38 Providing the additional source folder to the project - 1

5. Select the check box next to services folder (Figure 9-39).

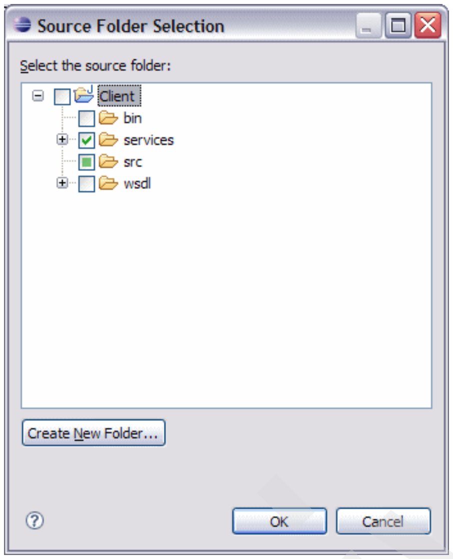
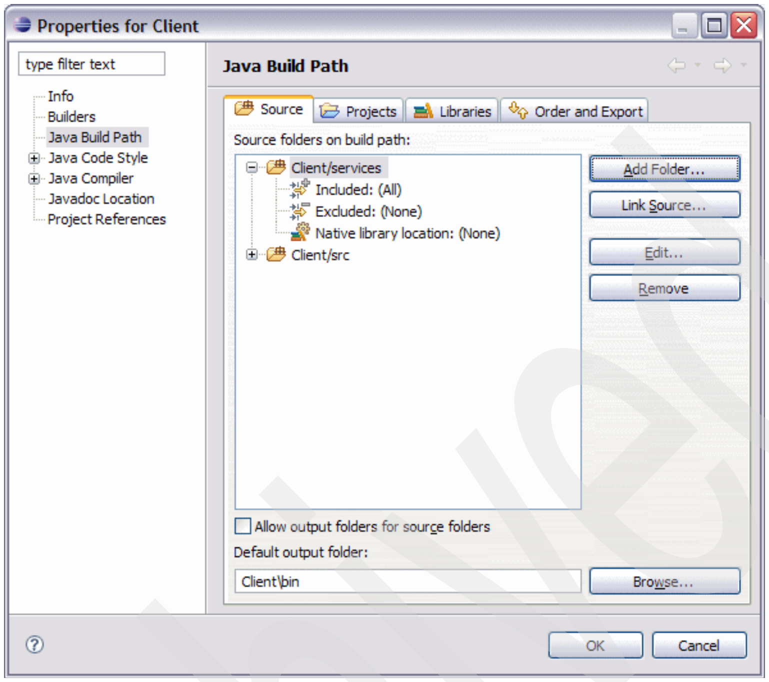


Figure 9-39 Providing the additional source folder to the project - 2

6. Click **OK** and leave the default inclusion and exclusion filters (Figure 9-40).



Source folders on build path:

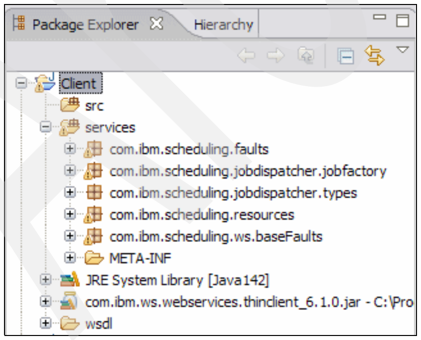
```
Client/services
    Included: (All)
    Excluded: (None)
    Native library location: (None)
Client/src
```

☐ Allow output folders for source folders

Default output folder: Client\bin

Figure 9-40 Providing the additional source folder to the project - 3

7. Click **OK**.
8. In the Package Explorer expand the **services** folder. You should see the Web services client packages added to the Client project (Figure 9-41).



```
Client
    src
    services
        com.ibm.scheduling.faults
        com.ibm.scheduling.jobdispatcher.jobfactory
        com.ibm.scheduling.jobdispatcher.types
        com.ibm.scheduling.resources
        com.ibm.scheduling.ws.baseFaults
        META-INF
    JRE System Library [Java142]
    com.ibm.ws.webservices.thinclient_6.1.0.jar - C:\Pro
    wsdl
```

Figure 9-41 Web Service client Java packages added to the project

Now you have finished the necessary steps to provide the project with the Web Service Java packages. These packages allow you to use a JobFactory

Web Service provided by the Tivoli Dynamic Workload Broker server from your client.

Adding the application logic

In this section we demonstrate how to create the package and the class that contain the main application logic. This core application class uses the Java packages generated in “Creating the Web Service client” on page 445.

We demonstrate the following tasks:

- Creating the package containing the core application class
- Importing the packages generated in the previous step into the core application class
- Providing the source code using the Web Service client Java packages generated in the previous step

Note: We use the term *core application class* in this scenario. This is just our terminology that helps us distinguish the class that implements the application logic from the classes that were generated from the WSDL definitions. *Core application class* is not an official Java term.

Creating the package and Java class

In this section we describe how to create the package with class that includes the application logic and references (imports) the Web Service client Java packages that we generated from the WSDL file.

1. First we create a new package. In the Package Explorer navigate to the source folder under our project. Expand **Client** and right-click **src**. Select **New** → **Package** (Figure 9-42).

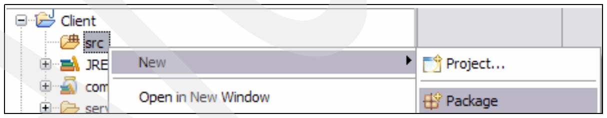


Figure 9-42 Creating a new package - 1

2. Type the package name. In this scenario we use com.ibm.scheduling.Submitter as the package name (Figure 9-43).

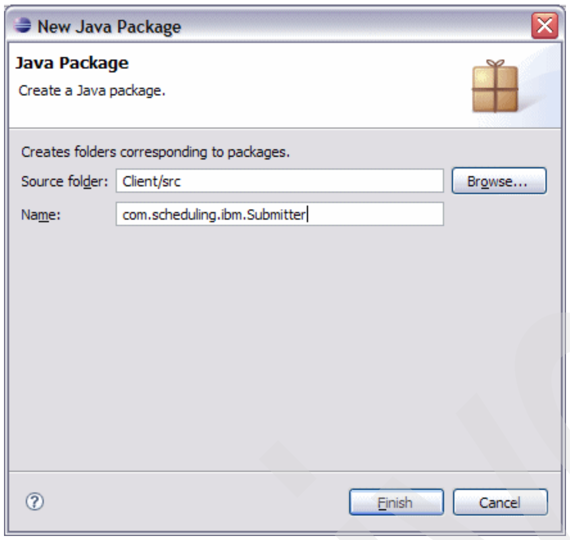

New Java Package
Java Package
Create a Java package.
Creates folders corresponding to packages.
Source folder: Client/src
Browse...
Name: com.scheduling.ibm.Submitter
Finish
Cancel

Figure 9-43 Creating a new package - 2

3. Click **Finish**. You should see the new package com.ibm.scheduling.Submitter in the Package Explorer. Right-click it and select **New** → **Class** (Figure 9-44).

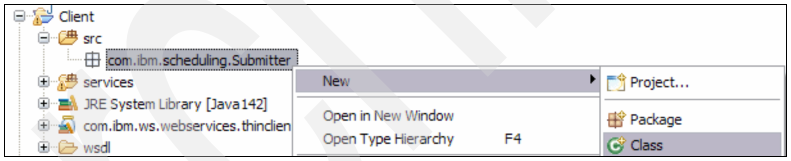
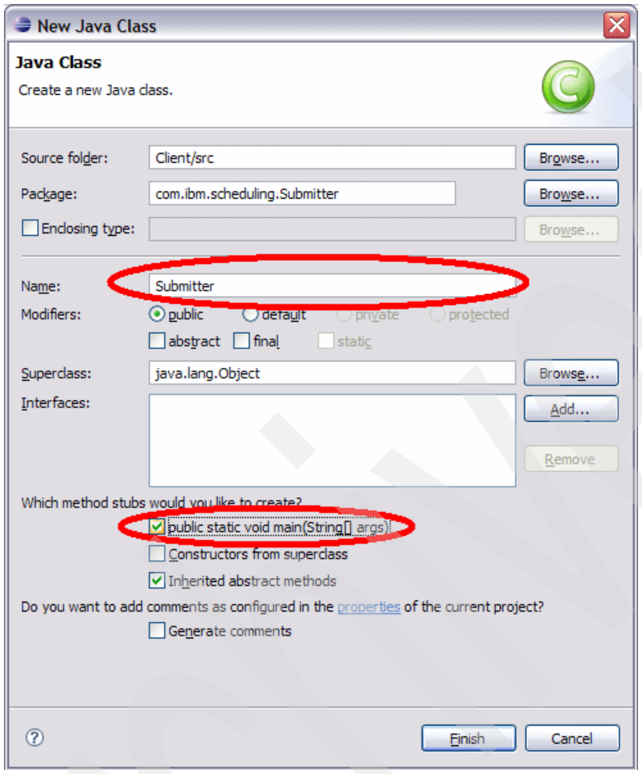

Client
src
com.ibm.scheduling.Submitter
services
JRE System Library [Java142]
com.ibm.ws.webservices.thinclien
wsdl
New
Open in New Window
Open Type Hierarchy F4
Project...
Package
Class

Figure 9-44 Creating a new class - 1

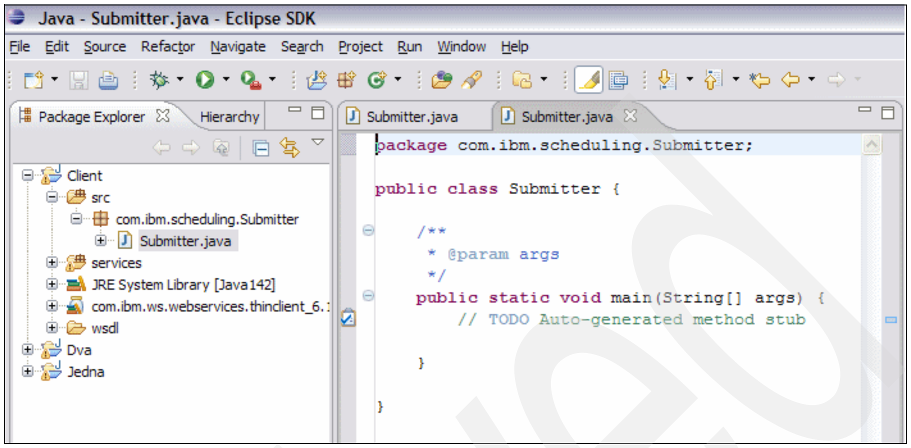
4. Type in the class name. In our scenario we use Submitter as the class name. Select the check box **public static void main(String[] args)** to include the template of the main method in the class definition (Figure 9-45).



Source folder: Client/src
Package: com.ibm.scheduling.Submitter
Enclosing type: [ ]
Name: Submitter
Modifiers: [✓ public] default private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
[✓] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods
Do you want to add comments as configured in the properties of the current project?
[ ] Generate comments

Figure 9-45 Creating a new class - 2

5. Click **Finish**. The generated class should appear in your workspace. Figure 9-46 shows the workspace with the generated class.


Java - Submitter.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Hierarchy
Submitter.java
Submitter.java
Client
src
com.ibm.scheduling.Submitter
Submitter.java
services
JRE System Library [Java142]
com.ibm.ws.webservices.thinclient_6.1
wsdl
Dva
Jedna

```
package com.ibm.scheduling.Submitter;

public class Submitter {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Figure 9-46 Workspace with the generated class

Importing the generated packages

In this section we describe which generated Web service client Java packages should be imported into the core application class. We list the packages that must be imported in order to use the JobFactory Web Service.

According to the namespace-to-package mapping that we specified by using the Scheduling-N2PMap.jd.properties file (while generating the packages from WSDL definitions), the package names begin with the com.ibm.scheduling.jobdispatcher prefix.

To allow the sample client to use classes defined within these packages, you must *import* the packages into the core application class that will reference them. Importing allows you to reference to the classes without referencing the package name. Example 9-36 shows which statements are used for importing the packages generated from the JobFactory.wsdl file.

Example 9-36 Importing the packages

```
import com.ibm.scheduling.jobdispatcher.jobfactory.JobFactory;
import
com.ibm.scheduling.jobdispatcher.jobfactory.JobFactoryServiceLocator;
import com.ibm.scheduling.jobdispatcher.jobfactory.JobStatus;
import com.ibm.scheduling.jobdispatcher.types.JobStateEnumeration;
```

The Client application source code

In this section we provide the whole source code of the client core application class. This is the Submitter class defined within the com.ibm.scheduling.Submitter package.

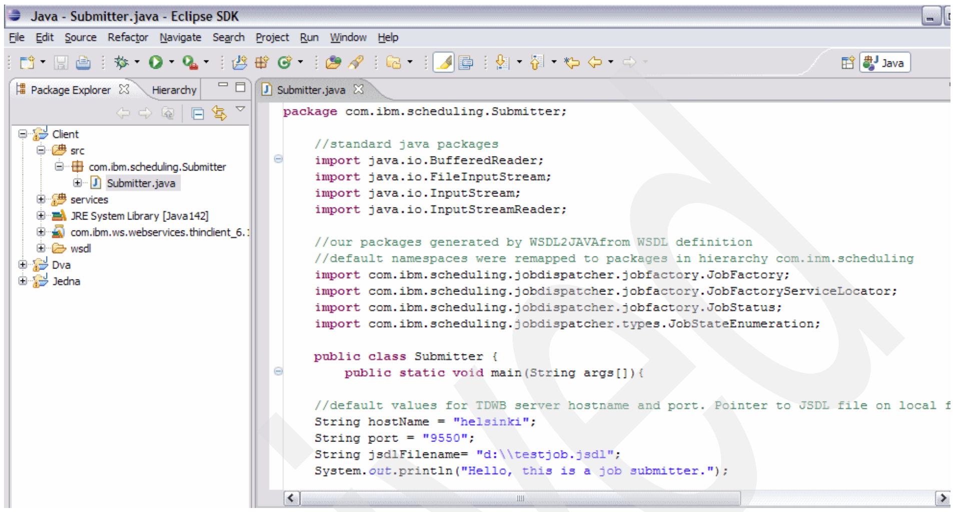
The whole code is included in Example 9-37. The code has the important parts fully commented.

Example 9-37 Core application class with main method

```
package com.ibm.scheduling.Submitter;

   //standard java packages
   import java.io.BufferedReader;
   import java.io.FileInputStream;
   import java.io.InputStream;
   import java.io.InputStreamReader;

   //our packages generated by WSDL2Java from WSDL definition
   //default namespaces were remapped to packages in hierarchy com.inm.scheduling
   import com.ibm.scheduling.jobdispatcher.jobfactory.JobFactory;
   import com.ibm.scheduling.jobdispatcher.jobfactory.JobFactoryServiceLocator;
   import com.ibm.scheduling.jobdispatcher.jobfactory.JobStatus;
   import com.ibm.scheduling.jobdispatcher.types.JobStateEnumeration;

   public class Submitter {
      public static void main(String args[]){

    //default values for TDWB server hostname and port. Pointer to JSDL file on local
filesystem.
   String hostName = "helsinki";
   String port = "9550";
   String jsdlFilename= "d:\\testjob.jsdl";
   System.out.println("Hello, this is a job submitter.");

   //instantiating new JobFactoryServiceLocator
   JobFactoryServiceLocator jfsl = new JobFactoryServiceLocator();

   //we will not catch each exception in this example
   try {
       //getting command line arguments. They can override hostname and port of TDWB server
and JSDL file name
      if (args.length > 0)
         hostName = args[0];
      if (args.length > 1)
         port = args[1];
      if (args.length > 2)
         jsdlFilename = args[2];
      System.out.println("Submitting job from JSDL file: "+jsdlFilename+". TDWB hostname: 
"+hostName+"TDWB port"+port);

       //reading the definition from JSDL file
       StringBuffer document = new StringBuffer();
       InputStream is = null;
       System.out.println("Reading jsdl file " + jsdlFilename);
       is = new FileInputStream(jsdlFilename);
       InputStreamReader isr = new InputStreamReader(is);
       BufferedReader br = new BufferedReader(isr);
       String line = null;
       while ((line = br.readLine()) != null) {
           document.append(line);
       } //while

       //creating URLs for JobFactory
      String jobFactoryURL = "http://" + hostName + ":" + port + 
"/JDServiceWS/services/JobFactory";

       //instantiating new jobFactory with corresponding URL
       JobFactory jf = jfsl.getJobFactory(new java.net.URL(jobFactoryURL));

      System.out.println("Submitting job");

      // submitting the job, using the JobFactory method submitJobFromJSDLXml()
      // getting the endpoint reference of that job to "epr". We will use "epr" later on,
      // whenever we will need to point to the instance of submitted job
      // operation submitJobFromJSDLXml requires several parameters, the only necessary for 
our purpose
      // is the JobDefinitionDocument (described in the type definition of 
submitJobFromJSDLXml operation)
      // the other 5 arguments are referrenced as null
      com.ibm.websphere.wsaddressing.EndpointReference epr = 
jf.submitJobFromJSDLXml(document.toString(),
            null,null,null,null,null);

      //printing EndpointReference to the submitted job
      System.out.println("Succesfull epr: " + epr);

      //instantiating an array of endpoint references
      com.ibm.websphere.wsaddressing.EndpointReference jobEprs[] = new 
com.ibm.websphere.wsaddressing.EndpointReference[1];

      //store the EndpointReference to the 1st position of the array
      //we need the array of Endpoint references for querying job states,
      //because the method getJobsStatuses() (a few lines below)
      //requires an array of EndpointReferences as its argument
      jobEprs[0] = epr;
      //beginning of loop
      boolean term = true;
      do {
         //querying the job status.
         //pass the ARRAY of endpoint references to the JobFactory method getJobsStatuses()
         //method getJobsStatuses() returns an array of job states as a response.
         //store the response into array "gjsResp" of type JobStatus
         JobStatus[] gjsResp = jf.getJobsStatuses(jobEprs);

         //if we have at least one job status in the array of job states
         if (gjsResp.length > 0) {
            //call the method getState() of JobStatus (instantiated to "gjsResp")
            //query the 1st position of the array. It corresponds to the 1st position
            //of our endpoint reference
            System.out.println("Job status: " + gjsResp[0].getState());
            //test, if there was not any fault while getting the job state
            if (gjsResp[0].getFault() == null) {
               //now compare the status of the jobs against the list of desired states
               //until the job that we have submitted, does not reach the desired state, the
loop will not end.
               term = (gjsResp[0].getState() == JobStateEnumeration.SUCCEEDED_EXECUTION) ||
               (gjsResp[0].getState() == JobStateEnumeration.FAILED_EXECUTION) ||
               (gjsResp[0].getState() == JobStateEnumeration.NOT_EXECUTED) ||
               (gjsResp[0].getState() == JobStateEnumeration.RESOURCE_ALLOCATION_FAILED);
               Thread.sleep(1000);
            } else {
               //if fault occured, print error code
               System.out.println("Error: " + gjsResp[0].getFault().getErrorCode());
               term = true;
            } //else (on getfault)
         } else {
            System.out.println("Error empty vector");
            term = true;
         } //else (on gjsResp.length)
      } while (!term);  //end of loop

      System.out.println("Succesfull");

      } catch (Exception e) {
         //we do not determine the exact exception...
      e.printStackTrace();
      } //catch
   } //main
} //Submitter
```

Paste this code into the Submitter.java class. Replace all of the previously generated content of the Submitter.java class, because the code provided in this example already contains all of the necessary content.

After putting this code into the Submitter.java, your workspace should look similar to Figure 9-47. We minimized some unimportant workspace windows because we want to show the most important windows in this figure.


Java - Submitter.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
Hierarchy
Client
src
com.ibm.scheduling.Submitter
Submitter.java
services
JRE System Library [Java142]
com.ibm.ws.webservices.thinclient_6.1
wsdl
Dva
Jedna
Submitter.java

```
package com.ibm.scheduling.Submitter;

    //standard java packages
    import java.io.BufferedReader;
    import java.io.FileInputStream;
    import java.io.InputStream;
    import java.io.InputStreamReader;

    //our packages generated by WSDL2JAVAfrom WSDL definition
    //default namespaces were remapped to packages in hierarchy com.inm.scheduling
    import com.ibm.scheduling.jobdispatcher.jobfactory.JobFactory;
    import com.ibm.scheduling.jobdispatcher.jobfactory.JobFactoryServiceLocator;
    import com.ibm.scheduling.jobdispatcher.jobfactory.JobStatus;
    import com.ibm.scheduling.jobdispatcher.types.JobStateEnumeration;

    public class Submitter {
        public static void main(String args[]){

    //default values for TDWB server hostname and port. Pointer to JSDL file on local f
    String hostName = "helsinki";
    String port = "9550";
    String jsdlFilename= "d:\\testjob.jsdl";
    System.out.println("Hello, this is a job submitter.");
```

Figure 9-47   Putting the code into Submitter.java

Providing the sample JSDL file

In order to be able to test the functionality of the sample client, you must point to the existing JSDL file located somewhere in your file system. You can create your own sample definition or use any existing JSDL file.

In Example 9-34 on page 461 we provide a sample JSDL definition that you can use for testing purposes. In our scenario this definition is stored in the D:\testjob.jsdl file.

Example 9-38   Sample JSDL definition

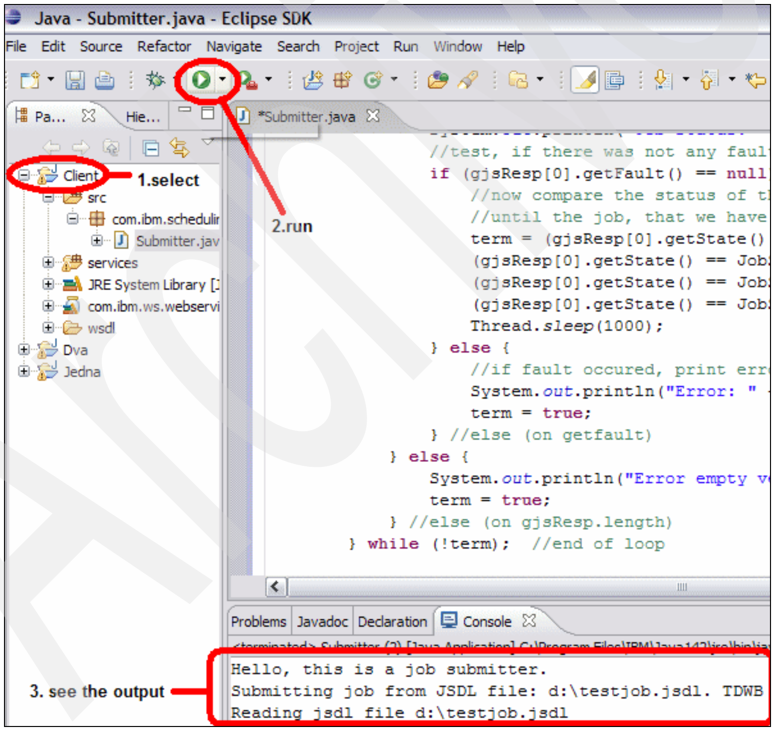
```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
```

```
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl JSDL.xsd
http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle JSDL-Native.xsd" description="Sample Test
job" name="testjob">
  <jsdl:annotation>This job is for testing purposes only. It does not require any special
resource to run.</jsdl:annotation>
  <jsdl:application name="executable">
    <jsdle:executable path="echo TDWB_SAMPLE_JOB"/>
  </jsdl:application>
</jsdl:jobDefinition>
```

Running the sample client from Eclipse

In this section we describe how to run the sample client from the Eclipse environment.

In the Package Explorer pane navigate to our project named Client. Then click the Run icon, as shown in the Figure 9-48. You should see output similar to that shown in the Figure 9-48.



```
            //test, if there was not any faul
            if (gjsResp[0].getFault() == null
                //now compare the status of t
                //until the job, that we have
                term = (gjsResp[0].getState()
                (gjsResp[0].getState() == Job
                (gjsResp[0].getState() == Job
                (gjsResp[0].getState() == Job
                Thread.sleep(1000);
            } else {
                //if fault occured, print err
                System.out.println("Error: " 
                term = true;
            } //else (on getfault)
        } else {
            System.out.println("Error empty v
            term = true;
        } //else (on gjsResp.length)
    } while (!term);  //end of loop
```

```
Hello, this is a job submitter.
Submitting job from JSDL file: d:\testjob.jsdl. TDWB
Reading jsdl file d:\testjob.jsdl
```

Figure 9-48   Running the sample client within Eclipse

For the instructions describing how to run the sample client from the command line see 9.5.4, “Running the sample client from the command line” on page 485.

9.5.4 Running the sample client from the command line

In this section we describe how to run the sample client from the command line.

To run the sample client from the command line, do the following:

1. Point to correct Java run time.
2. Point to the necessary JAR files.
3. Launch the core application class.

Example 9-39 shows the content of the script that includes all of the necessary statements for running the sample client from the command line.

Example 9-39 Script for running the sample client from the command line

```
set WAS_HOME=C:\Program Files\IBM\WebSphere\AppServer
set JAVA_HOME=C:\Program Files\IBM\WebSphere\AppServer\java

set WAS_CP="%WAS_HOME%\plugins\com.ibm.ws.runtimes_6.1.0.jar";
"%WAS_HOME%\runtimes\com.ibm.ws.webservices.thinclient_6.1.0.jar"

"%JAVA_HOME%\bin\java" -cp bin;%WAS_CP% com.ibm.scheduling.main.Submitter %1 %2 %3
```

In this example we use the Java run time provided with WebSphere Application Server V6.1. See 9.5.5, "Necessary Java run time and JAR files for running the client from the command line" on page 485, for more information about running the client from the command line.

Note: The example provided in this section is valid for Windows platforms. However, it can serve as a guide for UNIX/Linux platforms, too.

9.5.5 Necessary Java run time and JAR files for running the client from the command line

In this section we list the Java run time and JAR files that are necessary for running the client applications leveraging the Tivoli Dynamic Workload Broker Web services interface. We focus on the environment based on WebSphere Application Server V6.1.

The Java run time necessary for running the client is JDK 1.4.2 and later.

Each Web service provided by the Tivoli Dynamic Workload Broker server requires different JAR files.

According to Web Service used, you the need following JAR files to include in your classpath:

- JobFactory Web Service - %WAS_HOME%/runtimes/com.ibm.ws.webservices.thinclient_6.1.0.jar

  This JAR must be specified in the classpath.

- Job Definition Management Service - %WAS_HOME%/runtimes/com.ibm.ws.webservices.thinclient_6.1.0.jar

  This JAR must be specified in the classpath.

- Job Web Service

  This Web service is a special case. *Passing the JAR files just to the classpath is not sufficient.* You must use the special way of including the JAR files. See in Example 9-40 how the client can be run.

> **Note:** We launch an application named TestSubmitWithGetProperties in Example 9-40 on page 486, which is different from our sample client shown in 9.5, "Creating the sample client" on page 435. We do not provide the source code for this client in this book. We just want to emphasize the fact that the way of running the clients that leverage Job Factory Web Service and Job Web Service is different.

Example 9-40 Launching an application leveraging job Web service

```
call "C:\Program Files\IBM\WebSphere\AppServer\bin\setupCmdLine.bat"

set MYWAS=C:\Program Files\IBM\WebSphere\AppServer
set JAVA_HOME=C:\Program Files\IBM\WebSphere\AppServer\java

"%JAVA_HOME%\bin\java"
-Djava.security.auth.login.config="%app_server_root%\properties\wsjaas_client.conf"
-Djava.ext.dirs="%JAVA_HOME%\jre\lib\ext;%WAS_EXT_DIRS%;%WAS_HOME%\plugins;%WAS_HOME%\lib\WMQ\j
ava\lib" -Djava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory
-Dserver.root="%WAS_HOME%" -classpath "%WAS_CLASSPATH%;build;build\lib\SchedulerWSClient.jar"
com.ibm.scheduling.test.TestSubmitWithGetProperties %1 %2 %3
```

> **Note:** The example provided in this section is valid for Windows platforms. However, it can serve as a guide for UNIX/Linux platforms, too.

10

Troubleshooting

In this section we describe how to handle problem situations that you can face when installing, configuring, or operating Tivoli Dynamic Workload Broker.

We also include descriptions of how to solve the problems that can occur when integrating Tivoli Dynamic Workload Broker with other products. We provide hints, solutions, or workarounds where possible.

The following are covered in this chapter:

10.1 Troubleshooting the Tivoli Dynamic Workload Broker installation

Table 10-1 gives you information about the logs and trace files when troubleshooting the Tivoli Dynamic Workload Broker installation.

Table 10-1 Location of logs and trace files

Component	Path	Files	Content
Tivoli Dynamic Workoad Broker server	<WAS_profile_root> /default/logs/server1	native_stderr.log native_stdout.log serverStatus.log startServer.log stopServer.log SystemErr.log SystemOut.log trace.log	Tivoli Dynamic Workload Broker server log files Agent Manager log files
	tmp/TDWB	trace_installation.log msg_installation.log trace_installation_xml.log	Tivoli Dynamic Workload Broker server trace files. The trace_installation_xml.log is for use with troubleshooting tools.
	tmp	TDWBSilentResult.log	Tivoli Dynamic Workload Broker server silent installation result file
Tivoli Dynamic Workload Broker Web Console	<ISC installation directory> AppServer/profiles/default/logs/server1	native_stderr.log native_stdout.log serverStatus.log startServer.log stopServer.log SystemErr.log SystemOut.log trace.log	Web Console logs

Component	Path	Files	Content
ISC	<ISC>/AppServer/profiles/default/logs/ISC_Portal	startServer.log, stopServer.log	To troubleshoot unable to start/stop the ISC Note: If the PID file exists, this means that the ISC is up. If the ISC is down, but the file still exists, delete the file before restarting.
	<ISC>/PortalServer/log/	SystemOut.log trace.log SystemErr.log	SystemOut.log: Contains the TDWB log messages If trace is enabled, contains the TDWB trace and log messages SystemErr.log: Contains all Java exceptions Logging is always enabled. Tracing can be enabled using the ISC built-in panel found in the portfolio under “Console Settings”/”Enable Tracing
Command-line interface (CLI)	<ITDWB server installation directory>	CLItrace.log	To change the detail level for the cli trace file, change the setting in the property file.

Component	Path	Files	Content
Tivoli Dynamic Workload Broker agent	<ITDWB agent installation directory> /ep/logs	rcp.log	Agent logs and traces
	<ITDWB agent installation directory> /subagents/NativeJobExecutor/JM/<job id>	out.log, trace.log jm_exit.properties	Job execution logs and the exit status of the job
	tmp	TDWBSilentResult.log	Tivoli Dynamic Workload Broker server silent installation result file
	tmp/TDWB	trace_installation.log msg_installation.log trace_installation_xml.log	Tivoli Dynamic Workload Broker agent trace files The trace_installation_xml.log is for use with troubleshooting tools.
	<ITDWB agent installation directory> /ep/runtime/agent/logs	preinstall.log agentInstall.log uninstall.log nonstop.log	Intallation trace files of the Tivoli Common Agent
Tivoli Workload Scheduler agent	<ITDWB server installation directory> WebSphere/AppServer/profiles/default/logs/server1	trace.log	Tivoli Workload Scheduler agent log equivalent to the Tivoli Workload Scheduler standard agent log as saved in a stdlist
	tmp/TDWB	trace_installation.log msg_installation.log trace_installation_xml.log	Tivoli Workload Scheduler agent trace files The trace_installation_xml.log is for use with troubleshooting tools.

Note: The path to the profile depends on the profile that was chosen during the installation.

10.1.1 Tivoli Dynamic Workoad Broker Web Console and ISC logs

The following sections provide details of Tivoli Dynamic Workoad Broker Web Console and ISC installation logs.

ISC installation – install/uninstall log files

All the installation logs files are created in the $TEMP directory.

At install time, the following files are generated:

- tdwbwuinstall.log (install log of the ITDWB Web Console installer)
- ISCAction.log (ISMP log file for the ISCAction discovery task)
- ISCRuntimeInstall.log (ISMP log file for the ISCRuntime install task)
- ISCArchiveFixup.log (ant log for an ISCRuntime's subtask)
- ISCArchiveUpdatePortalPorts (ant log for an ISCRuntime's subtask)
- ISCPortalPostConfig.log (ant log for an ISCRuntime's subtask)
- ISCLogs_<date>_<time>.jar (bundle with all the uninstall logs)

At uninstall time, the following files are generated:

- tdwbwuuninstall.log (uninstall log of the ITDWB Web Console uninstaller)
- ISCRuntimeUninstall.log (ISMP log file for the ISC uninstall task)
- ISCLogBackup.log (ant log for the log backup subtask)
- ISCUninstallConfigTask.log (ant log for an ISC uninstaller's s subtask)
- ISCLogs_<date>_<time>.jar (bundle with all the uninstall logs)

Tivoli Dynamic Workoad Broker Job Brokering Definition Console

The Job Brokering Definition Console includes a logging and tracing facility. All message and trace logs are stored under the <user's home directory>\jd_workspace\.metadata directory.

Messages

The Job Brokering Definition Console messags are stored in the standard Eclipse log file .log. It also contains any errors Eclipse has captured and is very useful for debugging and FFDC.

Trace

Jlog is used for tracing. Trace files can be found under "tivoli\AWK\logs" in the .metadata directory by default. The trace logs also wrap after three log files reach a size of 1 MB each by default. Generally, if there is a problem the errors in the .log file will be sufficient. If there are not any error messages in the log or they are not specific enough, the trace will be necessary.

Tracing (configuration)

Tracing is configured through the Logging preference panel. The panel is shown by clicking **Window** → **Preferences** and selecting the **Logging** category. This table shows the trace logger, the trace level, and whether tracing is turned on. They can be configured by selecting the item in the table and selecting a new setting in the combo box. By default, tracing is set to maximum and is turned on.

Trace logs are configured through the Logging Output preference panel. The panel is shown by clicking **Window** → **Preferences** and selecting the **Output** category beneath the Logging category.

In the panel, the log file directory, log file size, and number of files used can be configured. Additionally, trace can be configured to go to the standard output console, to a file, or both.

Note: For additional install and uninstall logs related to the different components, refer to *IBM Tivoli Dynamic Workload Broker Installation and Configuration,* SC32-2282.

10.1.2 Activating traces for the Tivoli Dynamic Workload Broker server

To activate traces for the Tivoli Dynamic Workload Broker server:

1. Open a WebSphere Administrative console:

   ```
   http://<server ip address>:9060/ibm/console
   ```

   Where 9060 is the default port.
2. Go to Troubleshooting ?logs and then select **trace** and choose **server1**.
3. Go to Change log detail levels and then select configuration (persistent, needs restart) or run time (on running instance, not persistent).
4. In the details section, choose the component that you are interested in (for example, com.tivoli.agentManager, com.ibm.scheduling, TWSAgent) and the level of trace required for each (from off to all).
5. Restart the server1 application server as needed.

10.1.3 Activatinge traces for the Tivoli Dynamic Workload Broker Web Console

Do the following to activate traces for the Tivoli Dynamic Workload Broker Web Console:

1. Stop the ISC server:

   ```
   stopISC ISC_Portal
   ```

2. Edit the file:

   ```
   <ISC_installation_directory>\AppServer\profiles\default\config\cells
   \DefaultNode\nodes\DefaultNode\servers\ISC_Portal\server.xml.
   ```

3. Change the traceservice:TraceService definition as follows:

   ```
   <services xmi:type=?traceservice:TraceService?
   xmi:id=?TraceService_1132070385109? enable=?true?
   startupTraceSpecification=?com.ibm.scheduling.*=all: *=info?
   traceOutputType=?SPECIFIED_FILE? traceFormat=?BASIC?
   memoryBufferSize=?8?>
   ```

4. Restart the ISC server:

   ```
   startISC ISC_Portal
   ```

10.1.4 Diagnose failure dialogue - using the step list

If the installation fails, you can correct the error and then resume the installation using the step list window. This window allows you to run the installation steps one at a time, and lets you change the installation parameters and environment settings. This is a great benefit and helps reduce the number of failed installations. This benefit is at the expense of the automatic rollback of the installation provided by the installation wizard.

The Step List window can be displayed under two circumstances:

- If the Diagnose Failure window is displayed during an installation and you select the Diagnose failure radio button.
- You have previously tried to install Tivoli Dynamic Workload Broker, but the installation failed. You can resume the installation as follows:
  a. Make sure that CD1 is in the CD drive. Open a command-line interface.
  b. Change to the e:\TDWB directory, where e is the CD drive letter.

c. Run one of the following commands:
   - Windows:
     `setupwin32 -resume`
   - AIX:
     `setupaix -resume`
   - Linux:
     `setuplinux -resume`

The Step List window is displayed showing the installation steps. Steps that were successfully performed during the last installation are shown with their status set to success. Any steps that failed during the previous installation are shown with their status set to error.

The Step List window is organized as follows:

- Step #.
- Description.
- Target - The computer where the step is being installed.
- Status - The step status to one of the following:
  – Ready - The step is ready to be installed.
  – Success - The step has successfully completed.
  – Error - The step completed, but errors have been detected.
  – Held - One of the prerequisite steps has failed.
- Run next - Start the next step in the list that has a status set to *Ready*.
- Run all - Start in sequence, all the steps in the list that have a status set to Ready.
- Stop - Stops the installation. This button is enabled only if you are running more than one step, and you have clicked **Run All**. If you are running a single installation it step will complete, then the installation will stop and wait for your next instruction.
- Stop on error - Select Stop on error to halt the installation when an error occurs.
- Search by status - Select the status you want to view, then click **Search**. The step list displays the first step in the step list with the selected status.
- Status - The status of the installation engine is one of the following:
  – Waiting - User action is required.
  – Running - Installation of a step is in progress.
  – Stopping - After the current step, the installation engine will stop.
  – Searching - The installation engine is searching for product images.
- Details - For each step status, shows the number of steps in that status. Also displays the total number of steps. If any of the steps are not in the Success

state after the installation, or you require more information about each individual step, double-click the step to open the Step window.

The Step window

If you double-click a step in the Step List window, the Step window opens. It has three tabs: the Status tab, Properties tab, and Output tab.

Status tab

The Status tab shows the status of the installation step: Ready, Success, Error, or Held. You can change the status from *error* to *ready* if the condition that caused a step to fail has been removed.

Properties tab

The Properties tab gives the user parameters required by the step. These might be the parameters that you have input in the installation wizard, or values that the wizard has determined according to the logic of its operations. For example, in this tab the property DB2 Client Flag is an internal property determined by the wizard.

Output tab

The Output tab shows the output and any errors that occurred for the installation step, and also the commands that were performed by the installation. The Output tab contains the following entries:

- Time stamp - The time that the command was run.
- Return code - The return code for the operation. 0 = OK, < 8 = warning, >= 8 = error.
- DiagRecord - A unique point of failure identification. This can be quoted to IBM Software Support if you need to request assistance.
- Command - The command that failed.
- Command output - Any output from the command (such as a return code or an error message number).
- Error log - Shows a list of errors that occurred during the installation of the step. If errors occurred, examine the errors and then fix them before you try to change the state of the step to ready in the step status dialog. The procedure for correcting a step that has failed and resuming the installation is described in the next section.

Correcting a failed step and continuing the installation

Use the following procedure to correct a failed step and continue the installation:

1. Use the Output tab to determine what problem occurred.

2. Consult the sections in this guide that describe how to resolve problems found with the installation or the help for the error message that has been displayed.
3. If the solution to the problem requires you to change one of the values that you entered in the installation wizard, select the **Properties** tab and make the required changes. Then click **Apply**.
4. Click the **Status** tab, change the status to ready, then click **Apply**. The Step list is redisplayed.
5. If you want to run only the step that failed to ensure, for example, that the change you made has worked, click **Run Next**. This runs the first step in the step list (in step number order) with a status of ready. When the step finishes successfully, you can run the other steps in the installation in the same way (in sequence) or you can use Run All.
6. To resume the installation, click **Run All**. The wizard attempts to complete all outstanding steps, starting with the step that you have modified.

10.1.5 Tivoli Dynamic Workload Broker server troubleshooting

The Tivoli Dynamic Workload Broker server installer uses the Dynamic Failsafe Installation Framework (CMISMP) to avoid failures.

During the first installation phase, files are installed on the box and minimal configuration is performed. If anything fails in this phase, the installation is rolled back.

During the second installation phase, the installation steps are run. If anything fails, the user is prompted to diagnose the failure.

Server installation trace and logs

the Tivoli Dynamic Workload Broker server installation trace and logs are located in the following directories:

- /tmp/TDWB/msg_installation.log: This is the log file of the installation. This file is in the Problem Determination XML format and is intended to be used by customer and support.
- /tmp/TDWB/trace_installation.log: This is the trace file of the installation. This file is in plain text and is supposed to be used by support (L2/L3).
- /tmp/TDWB/trace_installation_xml.log: This is the trace file of the installation in the Problem Determination XML format. This file is intented to be used to feed a Problem Determination tool.
- /tmp/TDWBSilentResult.log: This file holds a message with the result of the silent install.

10.1.6 Tivoli Dynamic Workoad Broker agent installation troubleshooting

The Tivoli Dynamic Workload Broker agent installer uses the Dynamic Failsafe Installation Framework (CMISMP) to avoid failures.

During the first installation phase, files are installed on the box and minimal configuration is performed. If anything fails in this phase, the installation is rolled back.

During the second installation phase, the installation steps are run. If anything fails, the user is prompted to diagnose the failure.

The following give possible Tivoli Dynamic Workoad Broker agent (Tivoli Common Agent) installation failures:

- AWKSRI103E – 113E: The Installation of the Tivoli Common Agent failed message.

  An error is returned by the TCA installer. The message explains what happend in details.
- AWKSRI130E: The Tivoli Common Agent did not register to the Tivoli Agent Manager.

  The Tivoli Common Agent did not succeed in contacting the Agent Manager and downloading the certificates.
- AWKSRI131E: The Tivoli Common Agent is not ready to accept requests on port 9510.

  The TCA did not successfully authenticate to the AM (often because of time differences between the TCA and the AM).

There is a single solution for all three errors listed above. For all of the problems above, troubleshoot the TCA-AM connection and restart the TCA and wait for the port 9510 to be opened before completing the installation.

Note: 130E and 131E do not occur in disconnected installation.

/tmp/TDWB/msg_installation.log: This is the log file of the installation. This file is in the Problem Determination XML format and is intended to be used by customer and support.

/tmp/TDWB/trace_installation.log: This is the trace file of the installation. This file is in plain text and is supposed to be used by support (L2/L3).

/tmp/TDWB/trace_installation_xml.log: This is the trace file of the installation in the Problem Determination XML format. This file is intented to be used to feed a Problem Determination tool.

/tmp/TDWBSilentResult.log: This file holds a message with the result of the silent install.

10.1.7 JBDC installation troubleshooting

There is no special infrastructure for troubleshooting/fail-safe installation since it is very simple. In case of failure the installation is rolled back.

The installer does the following:

1. Copies the appropriate JRE™ on the file system.
2. Copies the JBDC image on the file system.
3. Creates a shortcut in the Start menu.
4. Associates the application to the jsdl file extension.

The trace file of the installation is /tmp/TDWB/trace_installation.log. This file is in plain text.

10.1.8 JBDC-specific problems

There are a couple of known problems with submitting a large number of jobs all at once if the job is defined as an embedded script. The XML code may look like Example 10-1.

Example 10-1   XML code

```
......
<jsdl:application name="executable">
    <jsdle:executable>
      <jsdle:script>ls -l</jsdle:script>
    </jsdle:executable>
  </jsdl:application>
......
```

Sometimes a-problem arises in which some jobs succeed and some fail on the same resource with either of the errors shown in Example 10-2.

Example 10-2   JBDC specific problems

```
The following error has been generated: Error creating job process.
The reason code is: 13.
Explanation: Permission denied
```

```
or
/tmp/NativeJobExecutorScripts/TDWB_341766610930cd032ebe2ebd72c17c5c.sh:
0403-006 Execute permission denied.
```

The workaround for this problem is to create a local script on the resources and change the job definition from *Script* to *Execution File* and specify the path to the script that exists on all of the eligible resources.

10.2 DB2 troubleshooting

Since Tivoli Dynamic Workload Broker V1.1 uses the services of DB2 (Oracle support will be available in Tivoli Dynamic Workload Broker V1.2), it is important to know the best practices for troubleshooting DB2.

10.2.1 Diagnostic tools

The following are some of the DB2 diagnostic tools that you can use.

db2diag.log

In DB2 Version 8, the primary log file intended for use by database and system administrators is the Administration Notification log. The db2diag.log file, however, is intended for use by DB2 customer support for troubleshooting purposes.

The db2diag tool serves to filter and format the volume of information available in the db2diag.log.

Example 1: filtering the db2diag.log by database name.

If there are several databases in the instance, and you wish to only see those messages that pertain to the database "SAMPLE", you can filter the db2diag.log as follows:

```
db2diag -g db=TDWB
```

Note: The database name must be specified in all capital letters.

Thus, you would only see db2diag.log records that contained "DB: TDWB", such as those shown in Example 10-3.

Example 10-3 db2diag.log records

```
2007-04-12-19.08.41.052739-300 I11356253C398      LEVEL: Event
```

```
PID     : 54434                TID  : 1          PROC : db2agent
(TDWB) 0
INSTANCE: db2inst1             NODE : 000        DB   : TDWB
APPHDL  : 0-284                APPID: G930CC98.C634.070413001041
FUNCTION: DB2 UDB, config/install, sqlfLogUpdateCfgParam, probe:20
CHANGE  : CFG DB TDWB: "MaxAppls" <automatic> From: "84" To: "93"
```

db2support

For collecting information for a DB2 problem, an important DB2 utility is db2support.

The db2support utility is designed to automatically collect all DB2 and system diagnostic information available. It also has an optional interactive question and answer session, which poses questions about the circumstances of your problem.

Using db2support avoids possible user errors, as you do not need to manually type commands such as "GET DATABASE CONFIGURATION FOR <database name>" or "LIST TABLESPACES SHOW DETAIL". Also, you do not require instructions on which commands to run or what files to collect. Therefore, information-gathering for problem determination is quicker.

10.2.2 Approach to troubleshooting DB2

Typically, if there is an error interacting with DB2 the evidence will be found in the WebSphere trace logs. These are the steps to take to identify the problem:

1. Verify the DB2 Error message to diagnose the error:

   `db2 ? XXXnnnnn`

   Where *XXXnnnnn* represents a valid message identifier. For example, db2 ? SQL1776N displays help about the SQL1776N message.

Example 10-4   SQL1776N message

```
bash-2.03$ db2 backup tdwb
SQL0104N  An unexpected token "tdwb" was found following "BACKUP".
Expected
tokens may include:  "DATABASE".  SQLSTATE=42601

bash-2.03$ db2 ? SQL0104N


SQL0104N An unexpected token "<token>" was found following
          "<text>".  Expected tokens may include:
```

```
          "<token-list>".

Explanation:

A syntax error in the SQL statement was detected at the specified
token following the text "<text>".  The "<text>" field indicates
the 20 characters of the SQL statement that preceded the token
that is not valid.

 As an aid to the programmer, a partial list of valid tokens is
provided in the SQLERRM field of the SQLCA as "<token-list>".
This list assumes the statement is correct to that point.

 The statement cannot be processed.

User Response:

Examine and correct the statement in the area of the specified
token.

 sqlcode :  -104

 sqlstate :  42601
```

2. To diagnose DB2-specific problems:
   - List applications:
     - db2 list applications
     - db2 list applications show details

       Displays the application program name, authorization ID (user name), application handle, application ID, and database name of all active database applications.
   - List locks of each application:
     - db2pd -db tdwb

       Dump all the PD information for the database.
     - db2pd -db twdb -locks show -app -tran –logs

       Dump detailed information for the database about locks, applications, and transaction.
     - db2 get snapshot for locks on tdwb

       Gets detailed information for locks.

- db2 select tableid, tabname from syscat.tables where tabschema=‘TDWBSCHEMA'

  List IDs of tables in the DB, useful to understand output of db2pd –locks show.

10.2.3 Sample DB2 troubleshooting scenario

Now we walk through a real-life Tivoli Dynamic Workload Broker troubleshooting scenario, involving DB2.

1. Using the Tivoli Dynamic Workload Broker server’s CLI tools, an error is returned, as seen in Example 10-5.

Example 10-5 Error when using the Tivoli Dynamic Workload Broker server’s CLI tools

```
bash-2.03# pwd
/opt/IBM/ITDWB/Server/bin
bash-2.03# ./jobquery.sh -status 0
Call Job Dispatcher to query jobs
AWKCLI056E Job Dispatcher - operation failed AWKJDE009I An error
occurred accessing the job repository database. See the exception log
for details..
bash-2.03# cd /opt/IBM/ITDWB/Server/logs
bash-2.03# vi CLItrace.log
```

2. The last two entries in the CLItrace.log are shown in Example 10-6.

Example 10-6 Last two entries in the CLItrace.log

```
Apr 25, 2007 10:38:34 AM com.ibm.scheduling.cli.jd.commands.JobQuery
execute
SEVERE: AWKCLI056E Job Dispatcher - operation failed AWKJDE009I An
error occurred accessing the job repository database. See the exception
log for details..

Apr 25, 2007 10:38:34 AM com.ibm.scheduling.cli.jd.commands.JobQuery
execute
FINE: JD exception - Operation Failed
com.ibm.scheduling.faults.OperationFailedFaultType
        at
com.ibm.scheduling.faults.OperationFailedFaultType_DeserProxy.convert(O
perationFailedFaultType_DeserProxy.java:17)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.ja
va:85)
```

3. Look at WebSphere logs, as in Example 10-7.

Example 10-7 WebSphere logs

```
bash-2.03# cd
/usr/IBM/WebSphere/AppServer/profiles/default/logs/server1
bash-2.03# vi SystemOut.log
```

4. Find the entry at the time of the CLItrace.log entries (Example 10-8).

Example 10-8 CLItrace.log entries

```
[4/25/07 10:38:33:550 CDT] 000002b1 DAO Logger    E   Exception:
                                 java.sql.SQLException: Connection
authorization failure occurred.  Reason: password invalid.DSRA0010E:
SQL St
ate = null, Error Code = -99,999DSRA0010E: SQL State = null, Error Code
= -99,999
```

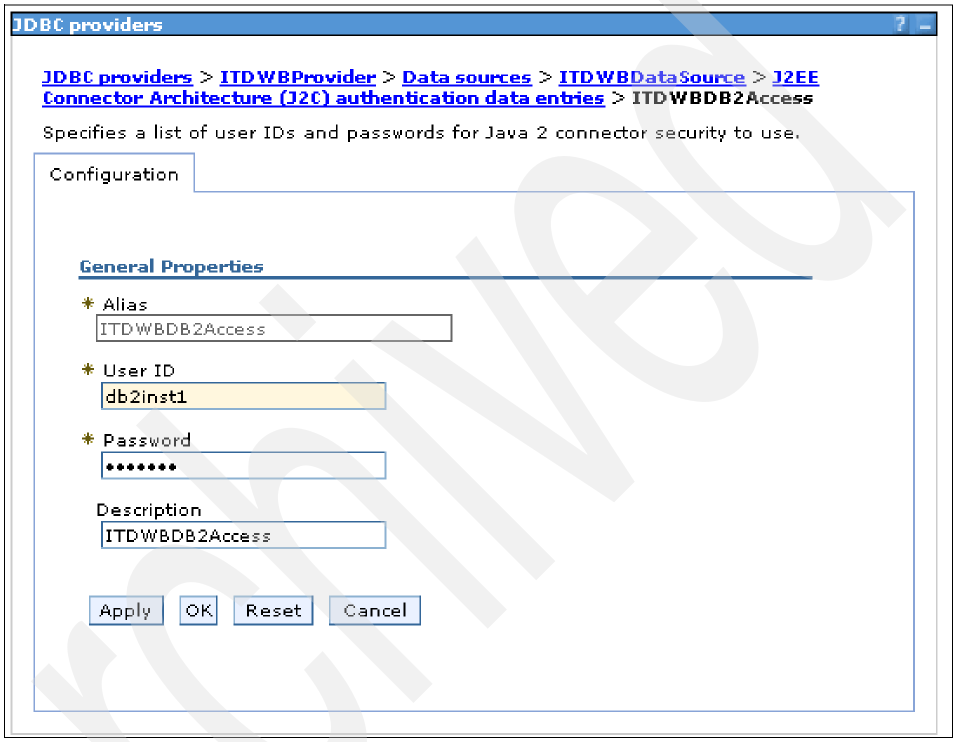
5. Test the db2inst1 user’s password (Example 10-9).

Example 10-9 Testing the db2inst1 user’s password

```
bash-2.03$ db2 connect to tdwb user db2inst1 using origpw
SQL30082N  Attempt to establish connection failed with security reason
"24"
("USERNAME AND/OR PASSWORD INVALID").  SQLSTATE=08001

bash-2.03$ db2 connect to tdwb user db2inst1 using newpwd

   Database Connection Information

 Database server        = DB2/6000 8.2.0
 SQL authorization ID   = DB2INST1
 Local database alias   = TDWB
```

Conclusion: The db2inst1 user's password was changed at the operating system level, but the WebSphere security was not updated with the change. In order to correct this problem the password either needs to be reset at the OS level or the WebSphere Admin Console needs to be used to update the password entries for db2inst1. Figure 10-1 gives an example of where to make these changes in WebSphere Admin Console:

`http://hostname:9061/admin`



JDBC providers > ITDWBProvider > Data sources > ITDWBDataSource > J2EE Connector Architecture (J2C) authentication data entries > ITDWBDB2Access

Specifies a list of user IDs and passwords for Java 2 connector security to use.

Configuration

General Properties

* Alias
ITDWBDB2Access

* User ID
db2inst1

* Password
•••••••

Description
ITDWBDB2Access

Apply | OK | Reset | Cancel

Figure 10-1 WebSphere Admin Console

In the WebSphere Admin Console the object path to AgentRegistryDBAuth is one place where the db2 user ID credentials need to be updated. See Figure 10-1.

10.3 Troubleshooting the integration with IBM Tivoli Monitoring

In this section we first provide you with information about the log and trace files for Tivoli Dynamic Workload Broker and Tivoli Monitoring (or Tivoli Enterprise

Portal or TEP) integration, and then we describe the known problems regarding this integration.

We also mention a few troubleshooting techniques on the Tivoli Monitoring side, mostly concerning the Universal Agent.

We do not focus on troubleshooting of Tivoli Monitoring itself, since it is not within the scope of this book. For more detailed information about troubleshooting Tivoli Monitoring refer to *Getting Started with IBM Tivoli Monitoring 6.1 on Distributed Environment,* SG24-7143, or to Tivoli Monitoring product documentation, such as *Tivoli Monitoring Installation and Setup Guide Version 6.1.0*, GC32-9407, or *IBM Tivoli Monitoring Administrator's Guide Version 6.1.0*, SC32-9408.

10.3.1 Log and trace files location

The TEP support logs and traces into the WAS log and trace file.

- TEP support log messages can be found in:

  ```
  <WAS_HOME>/profiles/<PROFILE>/logs/server1/SystemOut.log
  ```

- TEP support trace messages can be found in:

  ```
  <WAS_HOME>/profiles/<PROFILE>/logs/server1/trace.log
  ```

Log and trace can be enabled and modified by using the WAS Admin Console.

Component Message ID: TEP support log message ID is: AWKTEPxxxE.

10.3.2 Problems with running the integration script on Windows

When you launch the integration script on Windows, you can receive a message such as:

```
C:\Program’ is not recognized as an internal or external command,
operable program or batch file.
```

This message is caused by an error in the integration script. An internal defect 30542 has been opened and this problem will be fixed in future releases.

However, you can fix the problem by yourself. The cause of the problem is that the script uses variables that are substituted by a string containing C:\Program Files\IBM\Websphere, which includes the blank space. The command is interpreted only to the space and the rest is omitted. To fix this error, you must quote (“”) the occurrences of %WAS_HOME% and %CLASSPATH% variables.

To correct the error, do the following steps:

1. Open the integration script in any text editor. For the location of the integration script, see "Default values and file locations" on page 391.
2. Locate the first line of script. It should begin with this:

   ```
   @%WAS_HOME%\java\bin\java -classpath %CLASSPATH%
   ```

   The actual line is much longer. We show only the important part.
3. Add the quotes so that they surround the occurrences of the %WAS_HOME% and %CLASSPATH% variables. The beginning of the modified line should look like this:

   ```
   @"%WAS_HOME%\java\bin\java" -classpath "%CLASSPATH%"
   ```

   (And continue with the rest of the line.)

Now you should be able to run the integration script.

10.3.3 Wrongly interpreted characters in log file path on Windows

The value of the -eventFilePathName parameter can be unpredictably parsed on the Windows platform in Tivoli Dynamic Workload Broker 1.1. Some characters preceded by backslash (\) are wrongly interpreted as special characters. A typical example is \t, which is treated as TAB-sign.

If you are experiencing troubles with the value of the -eventFilePathName argument, you must use the double backslashes in the path. For instance, you should use C\\:\\this_path instead of C:\this_path.

Note: The double backslashes also precede the colon character.

An internal defect 30643 has been opened and this problem will be fixed in future releases.

10.3.4 Cannot specify multiple event types together with parameters

In this section we describe an error in the integration mechanism regarding specifying multiple event types together with overriding default values. Then we provide a workaround for this issue.

When launching the integration script and specifying multiple event types (using -events argument) together with additional parameters, such as -metafileName or -UAApname, the integration run time is not able to parse more than one event type. An integration script abends with an error message, as shown in Example 10-10.

An internal defect 31336 has been opened for this problem. This error affects only Tivoli Dynamic Workload Broker v1.1 and will be fixed in future releases.

Example 10-10 Integration script does not accept multiple event types

```
C:\PROGRA~1\IBM\ITDWB\Server\bin>tepconfig -UAInstDir C:\IBM\ITM
-eventFilePathName C\\:\\TEMP -metafileName c:\meta2.mdl -events SUCC
FAILE
D CANCEL -UAAppName NEW_APP
Apr 2, 2007 9:16:09 AM
com.ibm.scheduling.cli.tep.commands.TEPConfigurator scanInputParms
SEVERE: AWKTEP020E Incorrect input parameter -UAAppName specified.
```

Possible workarounds are as follows:

- Do not override other values, such as -UAAppName.
- Do not specify the event types while issuing the integration script. Edit the configuration file TEPListener.properties directly. For each event type append one new line with this content:

  `*eventType*=yes`

  A sample TEPListener.properties file is shown in the Example 10-11.

Example 10-11 Sample content of TEPListener.properties configuration file

```
#TEP Listener Configuration Properties
#Mon Apr 02 07:28:07 PDT 2007
EVENTFILEPATH=C\:\\TEMP
MAXEVTSIZE=20
UNKNOWN=yes
SUCC=yes
SUBMITTED=yes
FAILED=yes
```

For the location of the TEPListener.properties file see 8.4.13, “Default values and file locations” on page 391.

10.3.5 Cannot remove unwanted event types

In this section we describe an internal defect, which causes any additional run of the integration script to not delete unwanted event types from the TEPListener.properties configuration file.

When issuing the integration script for second (or any further) time, and redefining the list of event types, only new required event types are added, but the old unwanted event types are *not removed*. This is an internal error. An internal defect 31351 was opened for this problem. This error affects only Tivoli Dynamic Workload Broker v1.1 and will be fixed in future releases.

A possible workaround is to edit the TEPListener.properties file directly. For the the location of the TEPListener.properties file see 8.4.13, “Default values and file locations” on page 391.

Delete each unwanted event type by removing the whole row with this content:

```
unwantedEventType=yes
```

A sample TEPListener.properties file is shown in Example 10-11 on page 507.

10.3.6 Tivoli Dynamic Workload Broker log file not created

In this section we describe the possible causes of why the Tivoli Dynamic Workload Broker log file was not created after running the integration script.

Note: The log file does not get created until an defined event inside Tivoli Dynamic Workload Broker occurs. For instance, if you launched the integration script with the -event CANCEL parameter, only cancelled job instances will be reported to the log file. Until a job cancellation occurs a log file does not get created.

If Tivoli Dynamic Workload Broker processed several jobs and some of them were in the state that you selected for monitoring and the log file was still not created, do the following steps:

1. Make sure that you have recycled the WebSphere Application Server under which Tivoli Dynamic Workload Broker server runs. For determining the proper WebSphere Application Server instance on Windows, see “Recycling Tivoli Dynamic Workload Broker’s WebSphere Application server” on page 334.
2. Make sure that you are searching for the correct file in the correct directory.

   The name of the file looks like this:

   ```
   TEPEVENTyyyymmddHHMM.log
   ```

   For example, a log file that has been created on the 8th of March 2007 at 6:05 p.m. has following name:

   ```
   TEPEVENT200703081805.log
   ```

If you did not override the default path, check the log file presence in the default location. For a list of default locations see 8.4.13, “Default values and file locations” on page 391. Otherwise, search in the path that you have specified in the -eventFilePathName argument.

3. If you have overridden the default path by specifying the -eventFilePathName argument make sure that:

   – You did not include the file name in the path. You must not include the file name. The value of the -eventFilePathName argument must be only a path.

   – The path that you supplied to the -eventFilePathName argument existed *before* you recycled the Tivoli Dynamic Workload Broker’s WebSphere Application Server. If not, you must create the path and then recycle the WebSphere Application Server once more.

   – The path that you supplied to the -eventFilePathName argument on Windows uses double backslashes.

     Example 10-12 shows an example of WebSphere Application Server’s log file called SystemOut.log. The lines show that TEPListener component did not find the path. If you focus on the path that TEPListener is searching for, you can see that the path is not interpreted correctly.

     To avoid this problem, follow the steps described in 10.3.3, “Wrongly interpreted characters in log file path on Windows” on page 506.

Example 10-12 SystemOut.log of WebSphere Application Server - TEPListener

```
[3/7/07 10:07:16:609 PST] 0000003a TEPIntegratio I   AWKTEP013I TEP
listener has been configured to write events to C:logsdwb.log
[3/7/07 10:07:16:609 PST] 0000003a TEPIntegratio E   AWKTEP006E The
source file C:logsdwb.log could not be opened.
[3/7/07 10:07:16:609 PST] 0000003a TEPIntegratio E   AWKTEP004E Failed
to get an instance of the source file.
[3/7/07 10:07:16:766 PST] 00000039 TEPIntegratio E   AWKTEP004E Failed
to get an instance of the source file.
```

10.3.7 Application for log file monitoring is not visible in Tivoli Enterprise Portal (out-of-box integration)

In this section we describe the possible causes of why the application designated for monitoring the Tivoli Dynamic Workload Broker log file is not visible in the Tivoli Enterprise Portal.

First we describe the actions that are performed by the integration script.

The integration script does the following:

- Configures the TEPListener component (part of the Tivoli Dynamic Workload Broker server). TEPListener is responsible for logging the defined events into the log file.
- Imports a metafile into Tivoli Monitoring Universal Agent and thus defines a new application in the Tivoli Monitoringserver.

If you cannot see in Tivoli Enterprise Portal the application designated for Tivoli Dynamic Workload Broker log file monitoring, do the following steps:

- Check that there in no *refresh pending* in the Navigator pane in the Tivoli Enterprise Portal.
- Check whether the Tivoli Dynamic Workload Broker exists and lines are being added into it. See 10.3.6, "Tivoli Dynamic Workload Broker log file not created" on page 508, to for more details about Tivoli Dynamic Workload Broker log file.
- Make sure that the Universal Agent is able to use a FILE data provider as one of its data sources.

  Even if the Tivoli Monitoring Universal Agent should by default accept text log files as its data provider, we experienced different behavior in our scenarios. Unless we explicitly specified a "FILE" data provider in the configuration of Universal Agent, we did not see any input from the Tivoli Dynamic Workload Broker server.

  For detailed step-by-step instruction including snapshots, see 8.4.6, "Configuring Universal Agent to accept a FILE data provider" on page 337.

10.3.8 Application for custom script monitoring is not visible in Tivoli Enterprise Portal

In this section we describe the possible causes of why the application designated for monitoring the availability of the ITDWB enterprise application is not visible in the Tivoli Enterprise Portal.

If you have imported the metafile, instructing the Universal Agent to use a custom script for checking the availability of ITDWB enterprise application, and no new application has been added into Tivoli Enterprise Portal, do the following steps:

1. Check that there in no *refresh pending* in the Navigator pane in the Tivoli Enterprise Portal.
2. Make sure that the metafile points to an existing script. Check the value of SOURCE keyword in the metafile.

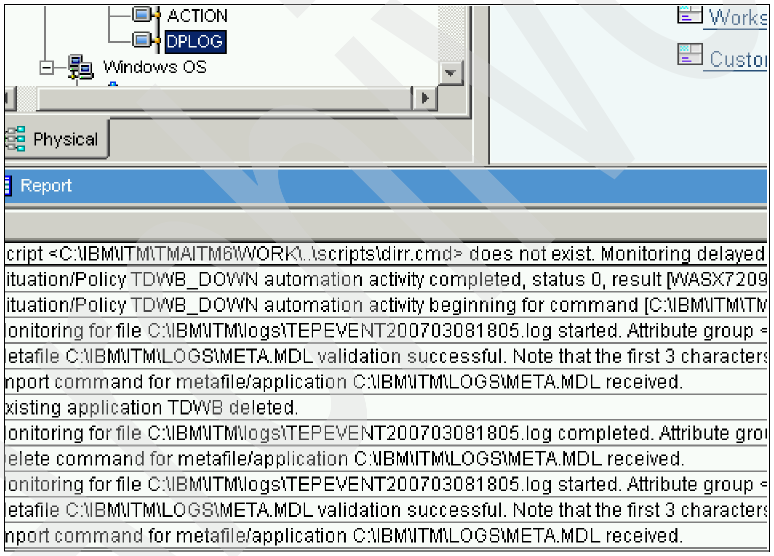
If the script does not exist or is located in another path, create it or copy into the proper location (corresponding to the value of the SOURCE keyword in the metafile). After that no additional actions are required. The Universal Agent discovers the script file by itself and starts collecting the data from the target system. See “Checking the log of Universal Agent’s data provider” on page 511 for instructions about how to check whether the universal agent is able to find the custom script.

Checking the log of Universal Agent’s data provider

In this section we provide a short example of how to look into the log of Universal Agent’s data provider.

Expand the branch of the **Universal Agent** on the corresponding server (Tivoli Dynamic Workload Broker server) and look into the DPLOG view. You may find useful messages about particular Universal Agent’s data providers.

Figure 10-2 shows the DPLOG view with the notification of a missing script file.



```
cript <C:\IBM\ITM\TMAITM6\WORK\..\scripts\dirr.cmd> does not exist. Monitoring delayed
ituation/Policy TDWB_DOWN automation activity completed, status 0, result [WASX7209
ituation/Policy TDWB_DOWN automation activity beginning for command [C:\IBM\ITM\TM
onitoring for file C:\IBM\ITM\logs\TEPEVENT200703081805.log started. Attribute group <
etafile C:\IBM\ITM\LOGS\META.MDL validation successful. Note that the first 3 characters
nport command for metafile/application C:\IBM\ITM\LOGS\META.MDL received.
xisting application TDWB deleted.
onitoring for file C:\IBM\ITM\logs\TEPEVENT200703081805.log completed. Attribute gro
elete command for metafile/application C:\IBM\ITM\LOGS\META.MDL received.
onitoring for file C:\IBM\ITM\logs\TEPEVENT200703081805.log started. Attribute group <
etafile C:\IBM\ITM\LOGS\META.MDL validation successful. Note that the first 3 characters
nport command for metafile/application C:\IBM\ITM\LOGS\META.MDL received.
```

Figure 10-2 Universal Agent’s DPLOG view - non existing script

10.4 Troubleshooting the integration with Enterprise Workload Manager

This section contains information about where to look when you have problems with the integration of the Tivoli Workload Scheduler and Enterprise Workload Manager.

10.4.1 Log and trace files location

The Tivoli Dynamic Workload Broker Enterprise Workload Manager Extension logs and traces into the WAS log and trace file.

- The log messages,can be found in:

  `<WAS_HOME>/profiles/<PROFILE>/logs/server1/SystemOut.log`
- The trace messages can be found in:

  `<WAS_HOME>/profiles/<PROFILE>/logs/server1/trace.log`

10.4.2 Log and trace enablement

Logs and traces can be enabled and modified by using the WAS Admin Console (EWLMBvc component).

Component Message ID: EWLM Extension log message ID is: AWKEWLxxxE.

10.5 Troubleshooting the integration with Tivoli Workload Scheduler

In this section we provide information about Tivoli Workload Scheduler integration troubleshooting and unsupported functions.

10.5.1 Log and trace files location

The Tivoli Workload Scheduler Agent logs and traces into the WAS log and trace file.

- The Tivoli Workload Scheduler Agent log messages can be found in:

  `<WAS_HOME>/profiles/<PROFILE>/logs/server1/SystemOut.log`

- Tivoli Workload Scheduler Agent trace messages can be found in:

  `<WAS_HOME>/profiles/<PROFILE>/logs/server1/trace.log`

Logs and traces can be enabled and modified by using WAS Admin Console (TWSAgent component).

Component Message ID: TWS Agent log message ID is: AWKTSAxxxE.

10.5.2 Debugging feature

It is possible to make Tivoli Workload Scheduler Agent trace more details about messages and events received and sent to the Tivoli Workload Scheduler, by enabling the DEBUG feature.

We need to add the TWS.Agent.Enable.Debug=true property into the TWSAgentConfig.properties file. (The TWSAgent application restart is required.)

10.5.3 Unsupported functions

The following Tivoli Workload Scheduler features are not supported by the Tivoli Workload Scheduler Agent:

- SSL communication with its parent
- Extended Agent options
- Switch fault tolerance
- Return code mapping
- Centralized scripts

Note: The Switch Manager function works, but the backup fault tolerance is not supported.

10.6 Troubleshooting the integration with CCMDB

In this section we provide information about where to look when you have problems with the integration of CCMDB and Tivoli Dynamic Workload Broker.

10.6.1 Log and trace files location

The serviceability of the ImportDataFromCMDB CLI follows the same rules as all of the other CLIs.

The CCMDB Cli logs and traces into the CLItrace.log file in the log directory under server installation directory (<TDWB_INSTALLATION_DIR>\log\CLItrace.log).

Log and trace can be enabled and modified by changing the CLIConfig.properties file.

Component Message ID: CCMDB CLI log message ID is: AWKCDMxxxE.

The CMDB API client has implemented its own log, and it is available under <TDWB_INSTALLATION_DIR>\log\CMDB.log.

Managing Tivoli Dynamic Workload Broker jobs using Tivoli Workload Scheduler for z/OS end-to-end

An enterprise that selects to integrate the Tivoli Dynamic Workload Broker with the Tivoli Workload Scheduler for z/OS end-to-end further expands and enhances the enterprise workload automation solution. Whether you are a planner, operator, administrator, systems programmer, or architect, you need an understanding of how this integration is done and how Tivoli Dynamic Workload Broker jobs are managed using Tivoli Workload Scheduler for z/OS end-to-end.

The following topics are covered in this chapter:

- “Monitoring and control” on page 602
- “Terminology” on page 622

11.1 Introduction

IBM Tivoli Workload Scheduler for z/OS provides advanced workload planning and choreography services, along with extensive calendar and event-triggering services. From a single point of control, it drives and controls the workload processing at both local and remote mainframe sites.

IBM Tivoli Workload Scheduler for z/OS end-to-end is an enterprise solution that combines the strengths of Tivoli Workload Scheduler for z/OS and Tivoli Workload Scheduler, allowing you to schedule and control jobs on mainframe, Windows, and UNIX environments, for truly distributed scheduling. Tivoli Workload Scheduler for z/OS is used as the planner for the job scheduling environment. Tivoli Workload Scheduler for z/OS controller and trackers are used to schedule on mainframe. Tivoli Workload Scheduler domain managers, standard, and fault-tolerant agents are used to schedule on the distributed platforms.

In the z/OS mainframe environment, execution of the Tivoli Workload Scheduler for z/OS scheduled jobs is managed by IBM Workload Manager for z/OS (WLM). Using WLM, the installation classifies the work running on the z/OS operating system in distinct service classes and defines goals for them that express the expectation of how the work should perform. WLM uses these goal definitions to manage the work across all systems of a sysplex environment.

IBM Tivoli Dynamic Workload Broker allows you to further extend Tivoli Workload Scheduler for z/OS end-to-end by matching and routing batch workloads to the best available resources in the distributed environment in an on demand manner. Dynamic brokering provides dynamic optimization of workload processing based on the performance of the scheduling infrastructure and workload demands.

Tivoli Dynamic Workload Broker integrates with the Tivoli Workload Scheduler for a z/OS end-to-end architecture to schedule, monitor, and manage Tivoli Dynamic Workload Broker jobs through the Job Scheduling Console or Tivoli Workload Scheduler for the z/OS ISPF dialog. The Tivoli Workload Scheduler for the z/OS end-to-end administrator can define in the Tivoli Workload Scheduler for z/OS end-to-end database job definitions and job streams to be assigned to computers or resources associated to the Tivoli Dynamic Workload Broker server.

11.1.1 Integration benefits

Tivoli Dynamic Workload Broker provides a series of improvements for distributed job scheduling on your existing Tivoli Workload Scheduler for z/OS end-to-end solution:

- Virtualization of the scheduling infrastructure by providing an abstraction layer on the resource selection
- Workload balancing by routing jobs among a group of resources according to the availability and activity levels of those resources
- SOA job brokering services
- Scheduling of IBM WebSphere Java 2 Enterprise Edition (J2EE) applications
- Automatic routing of jobs to the most appropriate resources based on job requirements
- Enhanced flexibility in workload distribution and running
- Automatic routing of jobs for which submission failed to appropriate resources

Tivoli Workload Scheduler for z/OS end-to-end also provides the following features to Tivoli Dynamic Workload Broker:

- End-to-end scheduling infrastructure
- Advanced scheduling, calendaring, planning, and choreographing capabilities

11.1.2 Terminology

Tivoli Workload Scheduler for z/OS and Tivoli Workload Scheduler are two somewhat different software programs, each with partially its own history and terminology. With Tivoli Dynamic Workload Broker entering the scene some new concepts and terminology also become part of your daily life. For this reason, there are sometimes two or more different and interchangeable names for the same thing. Other times, a term used in one context can have a different meaning in another context. The terms and acronyms that are used throughout this chapter are defined in 11.7, “Terminology” on page 622.

11.2 Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end architecture

In this section we discuss the Tivoli Workload Scheduler for z/OS end-to-end architecture, the Tivoli Dynamic Workload Broker architecture, and the ways that you can integrate them.

Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end architecture integration can be done in several ways. Either way, enterprise end-to-end scheduling is an integrated solution for workload scheduling in an environment that includes both mainframe and non-mainframe systems. End-to-end workload automation helps you manage and coordinate up to hundreds of thousands of workloads, executing the correct workload at the correct time and in the correct sequence.

Because end-to-end scheduling involves running programs on multiple platforms, it is important to understand how the different components work together when selecting the architecture.

It is also important to consider how the combination of processes, procedures, people, and end-to-end scheduling architecture helps you reach your business goals and service level agreements (SLAs). The benefits of this solution provides an efficiency to business processes for your enterprise. The results are increasing customer satisfaction, reducing or eliminating complaints from business partners, and improving the brand or company.

11.2.1 Tivoli Workload Scheduler for z/OS end-to-end scheduling

In the Tivoli Workload Scheduler for z/OS end-to-end scheduling network, a mainframe computer acts as the single point of control for job scheduling across the entire enterprise. Tivoli Workload Scheduler for z/OS is used as the planning central for the job scheduling environment. Tivoli Workload Scheduler fault-tolerant agents run work on the non-mainframe platforms, such as UNIX, Windows, and Linux.

Scheduling components

Components involved in Tivoli Workload Scheduler end-to-end scheduling are:

- Tivoli Workload Scheduler for z/OS *engine*

  The engine comprises two subcomponents, the *controller* and the *server.*

  – Tivoli Workload Scheduler for z/OS controller

    Manages database objects, creates plans with the workload, and executes and monitors the workload in the plan.

  – The Tivoli Workload Scheduler for z/OS server

    Acts as the Tivoli Workload Scheduler master domain manager. The Tivoli Workload Scheduler for z/OS server is the focal point for all communication to and from the Tivoli Workload Scheduler network.

- Tivoli Workload Scheduler *domain managers*, *fault-tolerant agents*, *standard agent* and *extended agents*

  Domain managers serve as communication hubs between Tivoli Workload Scheduler for z/OS and the fault-tolerant agents in each domain. Fault-tolerant agents are usually where the majority of distributed jobs are run.

  Standard agents and extended agents are used for special scheduling purposes. A standard agent is connected to a domain manager from where it receives the request and information to run a job.

Figure 11-1 shows a Tivoli Workload Scheduler network managed by a Tivoli Workload Scheduler for z/OS engine. This is accomplished by connecting Tivoli Workload Scheduler domain managers and Fault Tolerant Agents directly to the Tivoli Workload Scheduler for z/OS engine. The Tivoli Workload Scheduler for z/OS engine acts as the master domain manager of the Tivoli Workload Scheduler network.

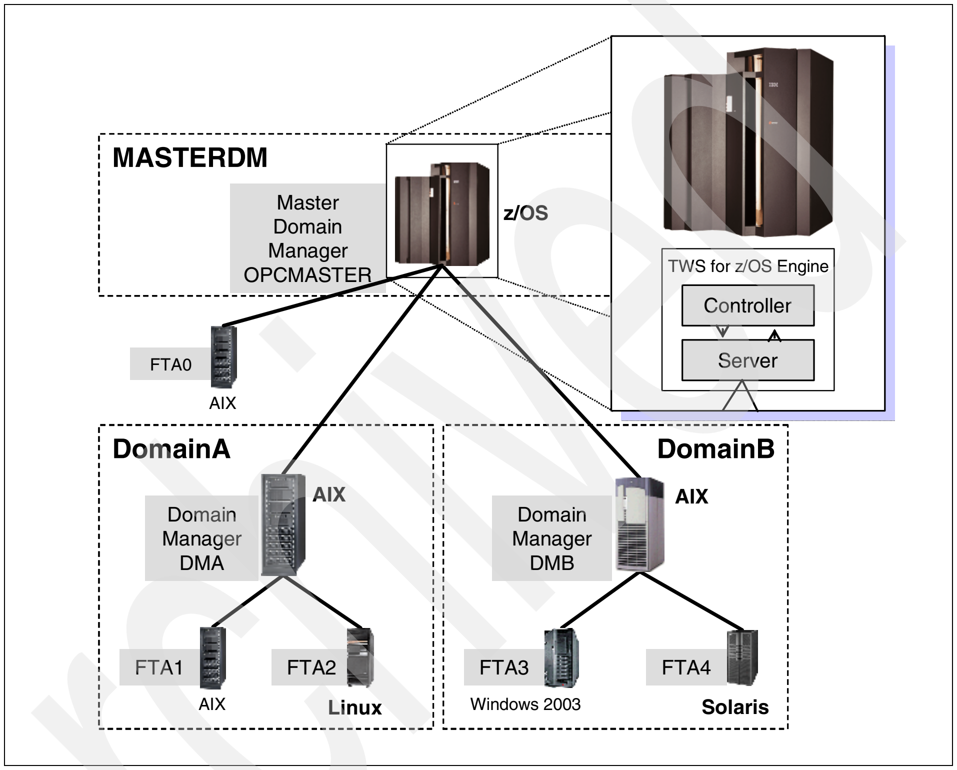


Figure 11-1 Tivoli Workload Scheduler for z/OS end-to-end scheduling architecture

User interfaces

Users interact with Tivoli Workload Scheduler for z/OS scheduling components using one or more of the following user interfaces:

- Job Scheduling Console

  Job Scheduling Console (JSC) is a common graphical user interface (GUI) to both the IBM Tivoli Workload Scheduler and IBM Tivoli Workload Scheduler for z/OS scheduling engines.

- Tivoli Dynamic Workload Console

  The Tivoli Dynamic Workload Console is a Web-based user interface for Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS.

- Tivoli Workload Scheduler for z/OS ISPF panels

  This user interface runs under Interactive System Productivity Facility (ISPF) on z/OS. It is available for Tivoli Workload Scheduler for z/OS.

Database and plan components

Key database and plan components in Tivoli Workload Scheduler for z/OS end-to-end are:

- Workstation database

  The workstation database contains Tivoli Workload Scheduler for z/OS controller definitions for workstations in both the mainframe and end-to-end scheduling network.

- Job stream database (Application Description database)

  The Job Stream database contains planning and choreography information for jobs in both the mainframe and end-to-end scheduling network.

- Long-term plan

  The long-term plan contains high-level job stream information for a period of up to four years.

- Current plan

  The current plan contains the detailed workload for both the mainframe and the end-to-end scheduling network. The current plan typically covers a period of 24 hours.

- Symphony file

  The symphony file contains the part of the workload that is defined to run on fault-tolerant workstations, standard agents, and extended agents. It also contains information about the end-to-end network topology and configuration.

Figure 11-2 is a high-level picture showing how database and plan components are related and used during current plan and symphony file creation.

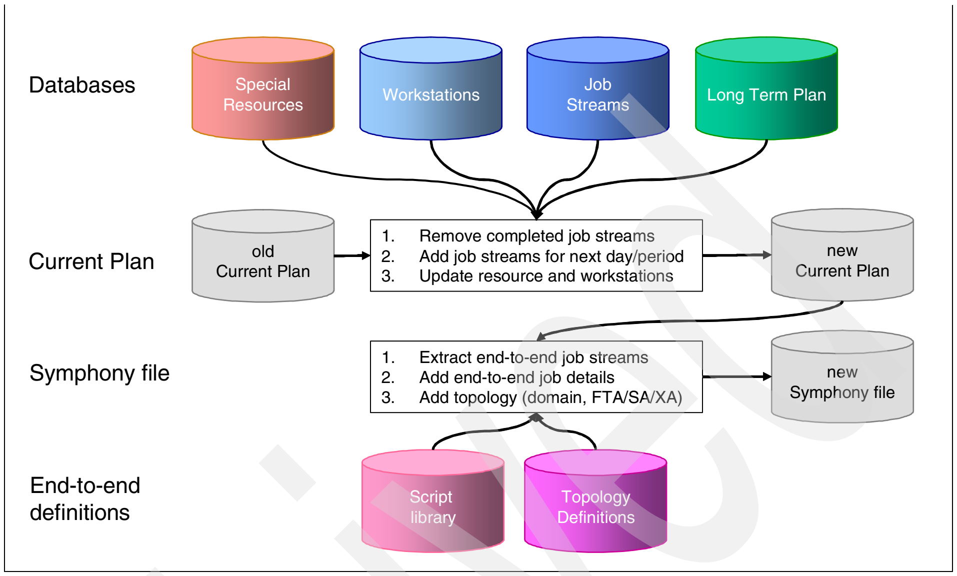

Databases
Special Resources
Workstations
Job Streams
Long Term Plan
Current Plan
old Current Plan
1. Remove completed job streams
2. Add job streams for next day/period
3. Update resource and workstations
new Current Plan
Symphony file
1. Extract end-to-end job streams
2. Add end-to-end job details
3. Add topology (domain, FTA/SA/XA)
new Symphony file
End-to-end definitions
Script library
Topology Definitions

Figure 11-2 Tivoli Workload Scheduler for z/OS end-to-end current plan and symphony file creation

Symphony file distribution

Tivoli Workload Scheduler for z/OS controller creates the current plan. It also creates the symphony file and sends it to the Tivoli Workload Scheduler for z/OS server.

The Tivoli Workload Scheduler for z/OS server, in its role as master domain manager, distributes the symphony file to the distributed network of domain managers and fault tolerant agents. Each subordinate domain manager in turn distributes the symphony file to its subordinate domain managers and fault-tolerant agents.

Standard agents receive a light version of the symphony file containing only domain, workstation, and user definitions.

Figure 11-3 shows how the symphony file is distributed to the Tivoli Workload Scheduler for the z/OS end-to-end network.

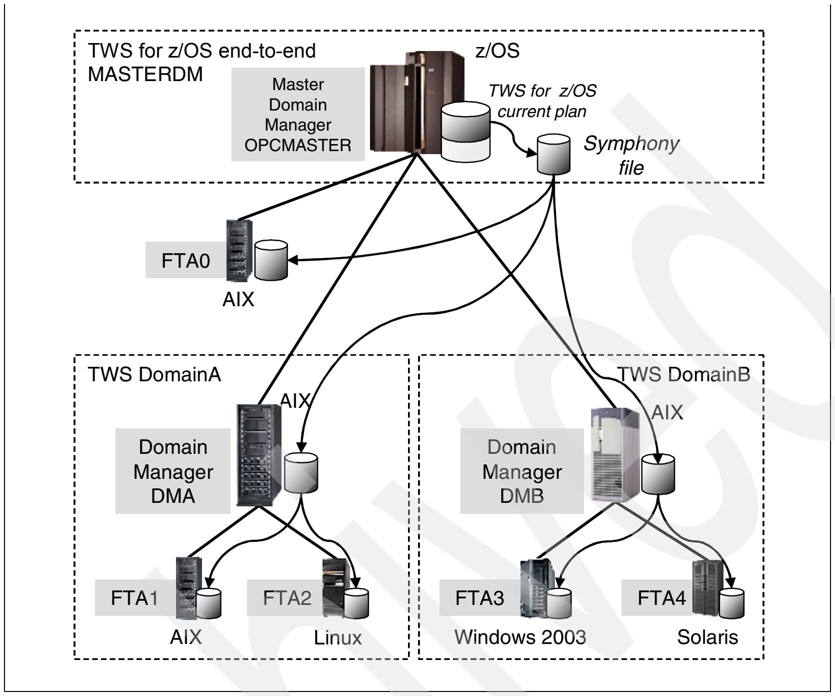


Figure 11-3 Symphony file distribution to Tivoli Workload Scheduler for z/OS end-to-end network

Planning and choreography

Scheduling analysts perform planning and choreography by interacting with Tivoli Workload Scheduler for z/OS controller using the available user interfaces.

Tivoli Workload Scheduler for z/OS provides extensive planning and choreography capabilities including calendaring and periods, definition of job stream run cycles in business terms, job dependencies, submit-on time requirements, and job resource requirements (using the special resource concept). Jobs are always defined as being part of job streams.

A key concept in Tivoli Workload Scheduler for z/OS is that every job, mainframe or distributed, is defined to run on one, and only one, workstation. For distributed jobs this has the consequence that changing the job to run on another computer implies that the job definition must be changed to use another workstation.

Job JCL and script definitions

Tivoli Workload Scheduler for z/OS end-to-end uses two types of definitions describing the job execution details:

- Job JCL definitions for mainframe jobs

  Mainframe job JCL definitions are defined using the z/OS Job Control Language (JCL). Job JCL definitions are called JCL members and are kept in z/OS data sets on mainframe.

  Each Tivoli Workload Scheduler for z/OS mainframe job references a corresponding job JCL member through the job name entity.

- Job script definitions for distributed jobs

  There are two types of distributed jobs in Tivoli Workload Scheduler for z/OS end-to-end, centralized script jobs and non-centralized script jobs:

  – Job script definitions for centralized script jobs

    Job script definitions for centralized script jobs are called centralized script members and are kept in z/OS data sets on the mainframe. Centralized script members contain the script source (for example, the UNIX script source).

    Each centralized script job references a corresponding centralized job script member through the job name entity.

    Copies of script definitions for centralized jobs are included in the current plan.

  – Job script definitions for non-centralized script jobs

    Job script definitions for non-centralized script jobs are called SCRPTLIB members and are kept in z/OS data sets on teh mainframe.

    Non-centralized script members contain a script or command name and, if needed, a path. The script source (for example, the UNIX script source) is kept on the distributed computer.

    Each non-centralized script job references a corresponding SCRPTLIB member through the job name entity.

    Copies of script definitions for non-centralized jobs are included in the symphony file.

Scheduling analysts:

- Maintain job JCL, centralized script members, and SCRPTLIB members by interacting with the z/OS ISPF user interface.
- Maintain the distributed script source by interacting with user interfaces on the distributed computers, such as Wordpad on Windows and VI editor on UNIX.

Job submission

Tivoli Workload Scheduler for z/OS end-to-end submission of mainframe jobs is controlled by job information in the current plan.

Submission of distributed jobs is controlled by job information in the current plan or the symphony file, depending on the type of job:

- When a centralized script type job is ready to run, the Tivoli Workload Scheduler for the z/OS end-to-end engine retrieves the corresponding script member and sends it to the fault-tolerant agent (FTA). The FTA submits the job to the operating system.
- When a non-centralized script type job is ready to run, the fault-tolerant agent retrieves the job script definition from the local symphony file and submits the job to the operating system.

Note: A Tivoli Workload Scheduler standard agent receives job submission requests from its domain manager. The domain manager retrieves the script definition and includes it in the job submission request.

Job tracking

Tivoli Workload Scheduler for z/OS end-to-end automatically tracks progress of the enterprise workload. Job progress information from both mainframe and distributed jobs are captured and sent to the Tivoli Workload Scheduler for z/OS controller in real time. The controller automatically maintains job stream and job status information in the current plan based on job progress information received.

Job progress information from distributed jobs is used by domain managers to maintain real-time job stream and job status information in local symphony files.

Monitoring and control

Operations analysts and scheduling analysts monitor and control workload progress and status by interacting with Tivoli Workload Scheduler for z/OS controller using the available user interfaces. Controlling the workload includes tasks such as adding job streams *ad hoc* to cthe urrent plan, job error handling, and performing job recovery.

Tivoli Workload Scheduler for z/OS provides extensive real-time capabilities for automatic handling and recovery of errors and outage situations, helping you minimize the impact on the enterprise workload.

11.2.2 Tivoli Dynamic Workload Broker scheduling architecture

In this section we briefly revisit concepts and components on the Tivoli Dynamic Workload Broker scheduling architecture that are key when discussing integration of the two architectures:

- Tivoli Dynamic Workload Broker server

  The Tivoli Dynamic Workload Broker architecture consists of a managing server, which manages its agents. The server interacts with users via its clients (user interfaces).

  The main purpose of the Tivoli Dynamic Workload Broker server is to determine the best fitting resource (that is, computer) for each job it is instructed to submit.

- Tivoli Workload Scheduler agent

  The Tivoli Workload Scheduler agent emulates a Tivoli Workload Scheduler standard agent. The Tivoli Workload Scheduler agent receives job launch requests from a Tivoli Workload Scheduler domain manager and forwards the requests to the Tivoli Dynamic Workload Broker server.

- Tivoli Dynamic Workload Broker agents

  An agent provides two major services: follows the instructions incoming from the server (such as a request to run a job), and notifies the server of hosting system utilization (such as CPU load and memory usage).

- Job Repository

  The Job Repository is managed by the Tivoli Dynamic Workload Broker server and keeps three types of data:

  – Job definitions

    Job definitions are stored in an XML type format using the Job Submission Description Language (JSDL).

  – Job instances

  – Historical data on job instances

- Job definitions and job instances

  Job definitions contain execution and resource usage characteristics for the jobs submitted by the Tivoli Dynamic Workload Broker server.

  When a job is submitted a job instance is created.

- Computer

  A computer in Tivoli Dynamic Workload Broker terminology is a place where jobs are executed. A computer is represented by the Tivoli Dynamic Workload Broker agent installed on it.

- Resource definitions

  In Tivoli Dynamic Workload Broker you can define two types of resources:

  – Logical resources

    Logical resources allow you to define a logical resource name and associate that name with one or several computers. A job requiring a logical resource is only eligible for submission to computers that have the logical resource associated.

  – Resource groups

    Resource groups are used to group computers, logical resources, or both. For example, a job requiring a computer resource group is only eligible for submission to computers that are members of the resource group.

- User interfaces

  There are three user interfaces:

  – Tivoli Dynamic Workload Broker Web Console

    A Web-based user interface for managing the Tivoli Dynamic Workload Broker environment. It allows you to define computers and resource definitions, edit job definitions, submit and monitor jobs, and recover failing jobs or resources.

  – Job Brokering Definition Console

    The Job Brokering Definition Console is a graphical tool that serves as a user-friendly interface for creating Tivoli Dynamic Workload Broker job definitions.

  – Command-line interface

    The command-line interface (CLI) allows the Tivoli Dynamic Workload Broker user to perform all of the essential operations necessary for scheduling and managing jobs.

Note: In this chapter the term *Tivoli Workload Scheduler agent* always refer to the Tivoli Dynamic Workload Broker component unless otherwise stated. In some of the figures Tivoli Workload Scheduler agent is abbreviated as TWS Agent.

The Tivoli Dynamic Workload Broker architecture is described in detail in Chapter 2, “Tivoli Dynamic Workload Broker architecture” on page 25.

11.2.3 Integrated Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end architecture

Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end are connected using the Tivoli Workload Scheduler agent. The Tivoli Workload Scheduler agent emulates a Tivoli Workload Scheduler standard agent and is connected to a Tivoli Workload Scheduler domain manager. It can also be connected to the Tivoli Workload Scheduler for z/OS end-to-end master domain manager.

Note: Starting with Tivoli Workload Scheduler for z/OS end-to-end Version 8.3, standard agents and fault-tolerant agents can connect directly to the Tivoli Workload Scheduler for z/OS end-to-end master domain manager (OPCMASTER). In previous releases of Tivoli Workload Scheduler for z/OS, end-to-end standard agents and fault-tolerant agents must connect using inter-positioned Tivoli Workload Scheduler domain managers.

The Tivoli Workload Scheduler agent and the Tivoli Dynamic Workload Broker server are both running on the same WebSphere instance, which means they are also running on the same physical server. Hence, choosing an integration architecture for Tivoli Workload Scheduler agent and Tivoli Workload Scheduler for z/OS end-to-end also means that you choose where you install your Tivoli Dynamic Workload Broker server.

Note: Tivoli Dynamic Workload Broker Version 1.1 only supports a single Tivoli Workload Scheduler agent.

Figure 11-4 shows an example where the Tivoli Dynamic Workload Broker server and Tivoli Workload Scheduler agent are running on an AIX machine. The Tivoli Workload Scheduler agent is connected to a Tivoli Workload Scheduler domain manager (in DomainA), which in turn is connected to the Tivoli Workload Scheduler for z/OS end-to-end master domain manager.

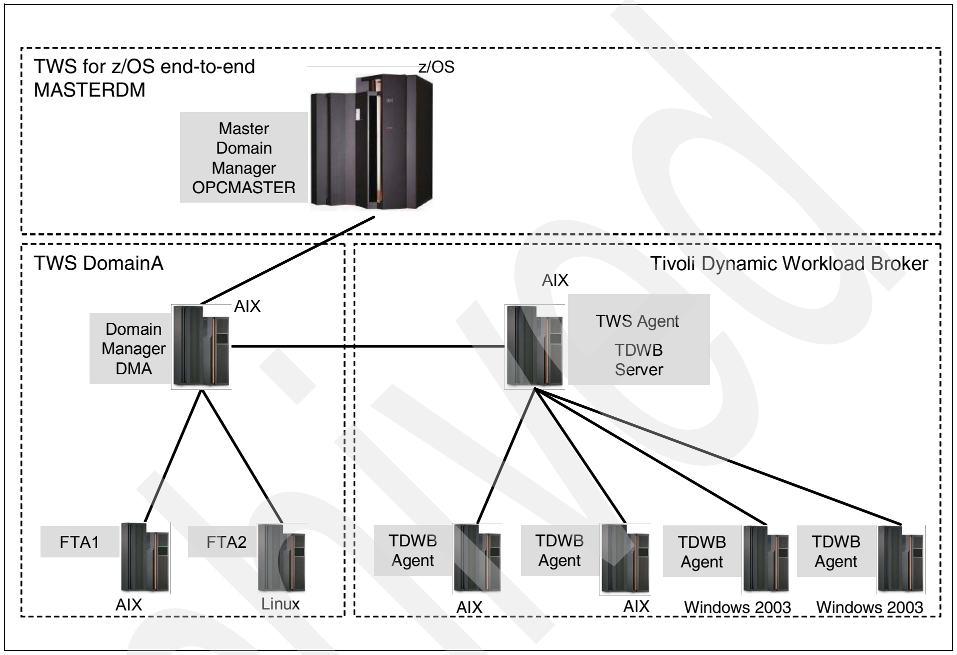


Figure 11-4 Tivoli Workload Scheduler agent connectivity example

Planning and Choreography integration

Integration allows you to combine Tivoli Workload Scheduler for z/OS end-to-end planning and choreography with the Tivoli Dynamic Workload Broker capability to dynamically select a target resource for job submission among a group of computers.

The Tivoli Workload Scheduler for z/OS current plan contains all jobs to be scheduled. When Tivoli Workload Scheduler for z/OS end-to-end and Tivoli Dynamic Workload Broker are integrated the current plan includes:

- Jobs to be run on mainframe servers. These jobs are defined on Tivoli Workload Scheduler for z/OS computer type workstations representing the target z/OS servers.
- Jobs to be run on distributed servers. These jobs are defined on Tivoli Workload Scheduler for z/OS end-to-end fault-tolerant workstations representing the target distributed servers.

- Jobs to be run by Tivoli Dynamic Workload Broker. These jobs are defined on a Tivoli Workload Scheduler for z/OS end-to-end workstation representing the Tivoli Workload Scheduler agent.

Figure 11-5 shows an example of a Tivoli Workload Scheduler for z/OS end-to-end job stream with five jobs:

- Job1 is a z/OS job defined to run on workstation CPU1. It is submitted by the Tivoli Workload Scheduler for z/OS for execution on z/OS.
- Job2 is an AIX job defined to run on workstation FTA1. It is submitted by the Tivoli Workload Scheduler fault-tolerant agent FTA1 for execution on AIX.

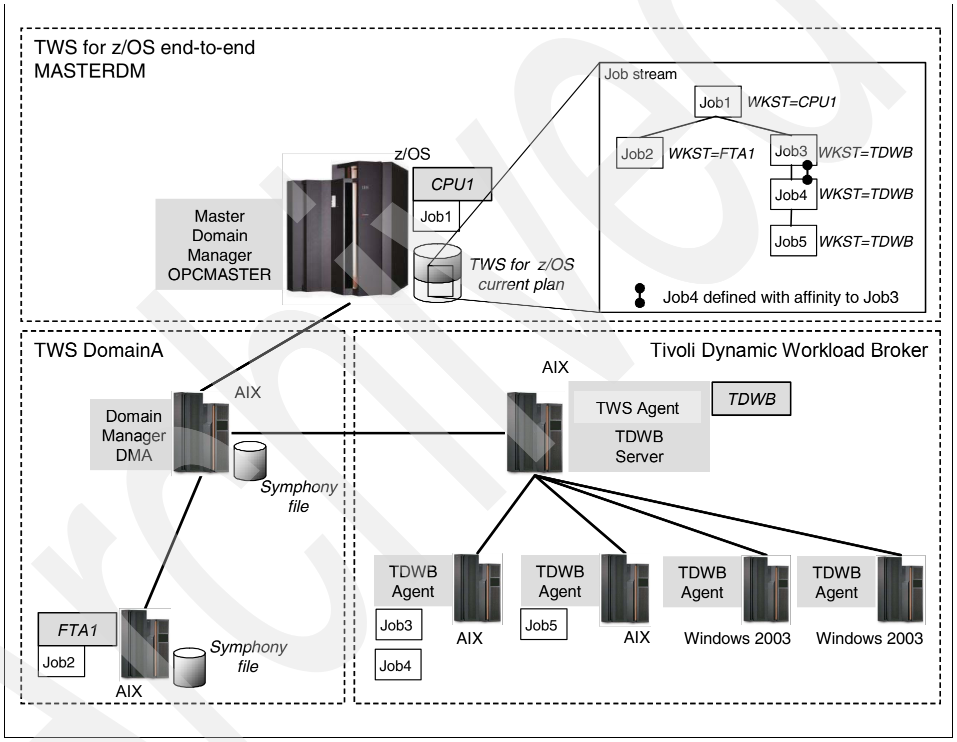


Figure 11-5 Job stream with z/OS, AIX, and Tivoli Dynamic Workload Broker jobs

- Job3 is a job defined to run on workstation TDWB.

  Domain manager DMA:

  – Retrieves the job3 definition from the symphony file. This includes the name of the Tivoli Dynamic Workload Broker job definition to be launched.

  - Sends a job launch request to the Tivoli Workload Scheduler agent, which in turn transfers the job launch request to the Tivoli Dynamic Workload Broker server.

  The Tivoli Dynamic Workload Broker server:

  - Retrieves the JSDL definition corresponding to Job3.
  - Allocates resources to Job3, including a target computer. (Logical and physical resource requirements for Job3 as well as current resource usage data are used to find a best-fit resource.)
  - Submits a job request to its agent on the selected AIX machine.
  - Waits for the execution result and when it arrives forwards it to the Tivoli Workload Scheduler agent, which in turn forwards it to domain manager DMA.

- Job4 is a job defined to run on workstation TDWB, including an affinity definition specifying that Job4 must run on the same resource as Job3 did. A job launch request is sent by domain manager DMA to the Tivoli Workload Scheduler agent, which in turn transfers the job launch request to the Tivoli Dynamic Workload Broker server.

  The Tivoli Dynamic Workload Broker server:

  - Retrieves the JSDL definition corresponding to Job4 and submits it to the agent on the same AIX machine that executed Job3
  - Waits for execution result and when it arrives forwards it to the Tivoli Workload Scheduler agent, which in turn forwards it to domain manager DMA

- Job5 is a job defined to run on workstation TDWB. A job launch request is sent by Tivoli Workload Scheduler domain manager DMA to the Tivoli Workload Scheduler agent, which in turn transfers the job launch request to the Tivoli Dynamic Workload Broker server.

  Since Job5 is not defined with any affinity, the Tivoli Dynamic Workload Broker server performs the same actions as described for Job3. But this time another AIX machine is selected as optimal for execution.

11.2.4 Examples of integration architectures

Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end can be integrated in several ways. We provide you with some typical examples along with key characteristics. The purpose is to aid you in choosing an architecture that fits into your environment and meets your needs and requirements.

Distributed Tivoli Dynamic Workload Broker directly connected to Tivoli Workload Scheduler for z/OS server

Figure 11-6 shows an integration where:

- The Tivoli Dynamic Workload Broker server is running on a distributed Windows 2003 server with its Tivoli Workload Scheduler agent connected directly to the Tivoli Workload Scheduler for z/OS V8.3 server.
- The existing distributed network, DomainA, is separated from the workload broker environment.

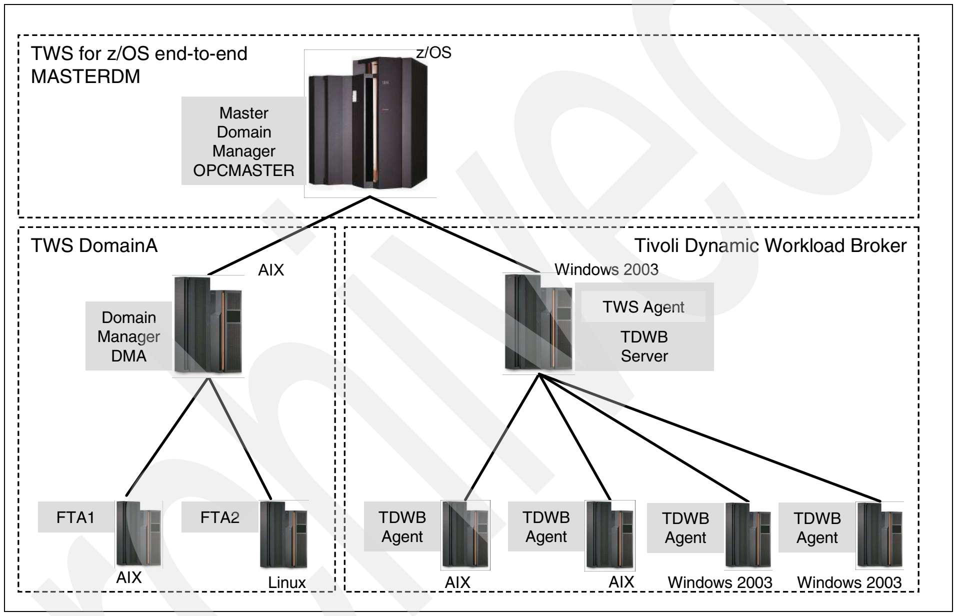


Figure 11-6 Distributed Tivoli Dynamic Workload Broker directly connected to Tivoli Workload Scheduler for z/OS server

Distributed Tivoli Dynamic Workload Broker connected to interpositioned Tivoli Workload Scheduler domain manager

Figure 11-7 shows an integration where:

- The Tivoli Dynamic Workload Broker server is running on a distributed AIX server with its Tivoli Workload Scheduler agent connected to interpositioned Tivoli Workload Scheduler domain manager DMA.
- This is a type of architecture that must be used if Tivoli Workload Scheduler for z/OS end-to-end is Version 8.2 or earlier.
- Job scheduling on computers managed by the Tivoli Dynamic Workload Broker requires domain manager DMA to be up and running. Domain DMA and the broker network are used for the same business area with the same availability requirements.

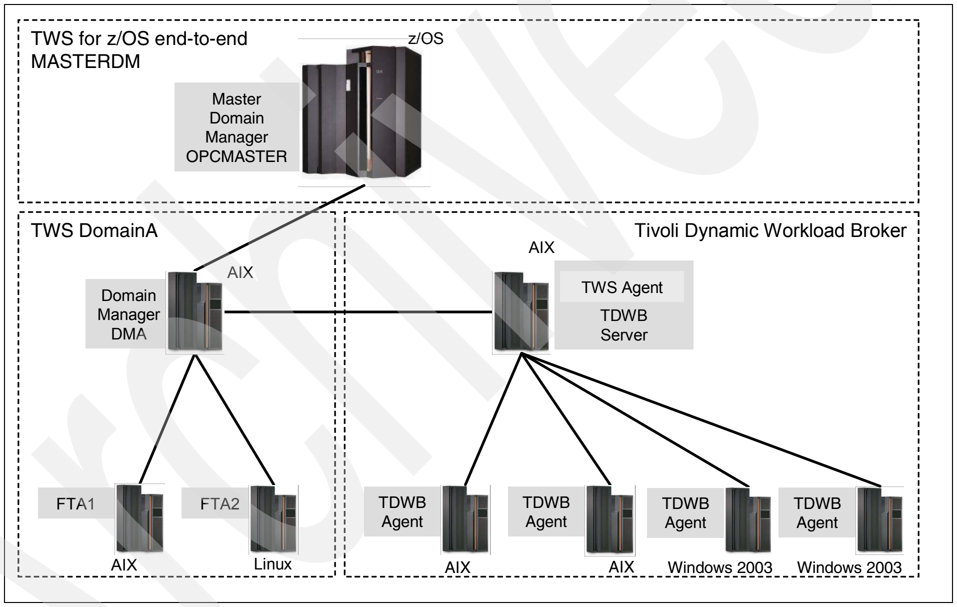


Figure 11-7 Distributed Tivoli Dynamic Workload Broker connected to interpositioned Tivoli Workload Scheduler domain manager

Tivoli Dynamic Workload Broker on zLinux directly connected to Tivoli Workload Scheduler for z/OS server

Figure 11-8 shows an integration where:

- The Tivoli Dynamic Workload Broker server is running on zLinux.
- Tivoli Workload Scheduler agent connects directly to the Tivoli Workload Scheduler for z/OS server.
- An existing AIX server farm managed by the Tivoli Dynamic Workload Broker is being migrated to zLinux. zLinux is implemented with the Tivoli Workload Scheduler agent installed.
- The migration of workloads from AIX to zLinux is done seamlessly using the capabilities of Tivoli Dynamic Workload Broker.

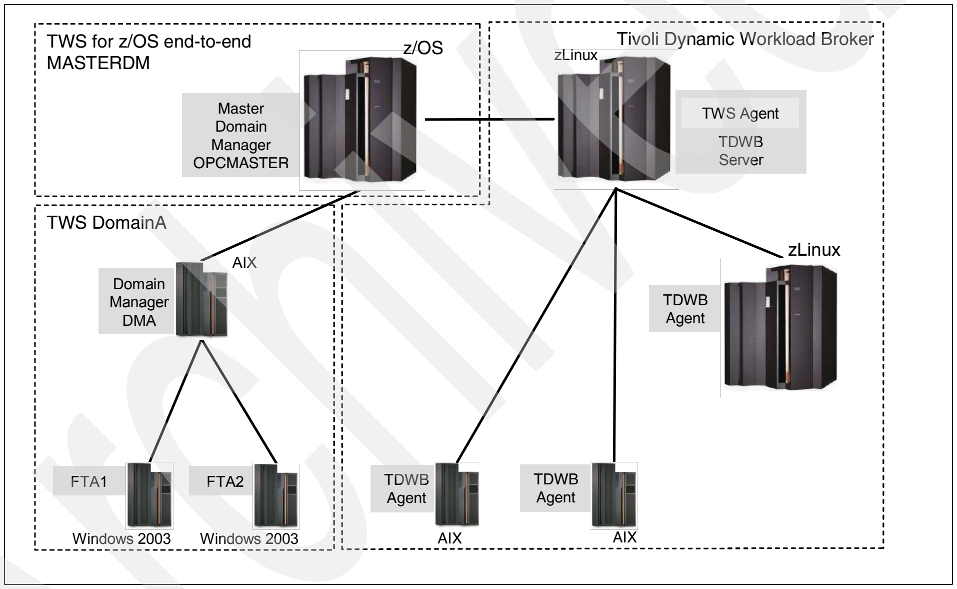


Figure 11-8 Tivoli Dynamic Workload Broker on zLinux

11.2.5 High availability and recovery integration

IBM Tivoli Workload Scheduler for z/OS end-to-end, IBM Tivoli Workload Scheduler, and Tivoli Dynamic Workload Broker are all well suited for environments that require high availability and short recovery times.

The integration architecture is used to identify which components need high availability implementation:

- Implementing high availability for the Tivoli Workload Scheduler for z/OS engine (Both controller and server components must be considered.)
- Implementing high availability for the Tivoli Workload Scheduler domain manager hosting the Tivoli Workload Scheduler agent

  This is necessary if the Tivoli Workload Scheduler agent is connected to an interpositioned domain manager instead of directly to the master domain manager.
- Implementing high availability for the Tivoli Dynamic Workload Broker server and Tivoli Workload Scheduler agent

Tivoli Workload Scheduler for z/OS engine high availability

The Tivoli Workload Scheduler for z/OS standby engine feature exploits the z/OS sysplex to deliver automatic, semi-automatic, or manual takeover from the active engine to the standby engine should an outage impact the active engine. One or more standby engines can be running at the same time, typically on different z/OS images in sysplex.

The Tivoli Workload Scheduler for z/OS end-to-end server allows you to exploit the Dynamic Virtual IP Address (DVIPA), allowing seamless connection switching to occur when a standby engine takes over the active engine role.

Tivoli Workload Scheduler domain manager high availability

Tivoli Workload Scheduler domain manager high availability is necessary if the Tivoli Workload Scheduler agent is connected to an interpositioned domain manager instead of directly to the master domain manager.

Tivoli Workload Scheduler domain manager high availability can be achieved using several options:

- IBM HACMP™

  IBM HACMP software is the IBM tool for building UNIX-based, mission-critical computing platforms. HACMP has two major components, high availability (HA) and cluster multi-processing (CMP).
- Microsoft® Cluster Service

  Microsoft Cluster Service (MSCS) is software that supports the connection of two servers into a cluster for higher availability and easier manageability of data and applications.

Tivoli Dynamic Workload Broker server high availability

Tivoli Dynamic Workload Broker server high availability is described in detail in Chapter 6, "High availability and recovery considerations" on page 247.

Tivoli System Automation for Multiplatforms

Tivoli System Automation is a high-availability product that strives to provide the continuous operation of a system over time. It uses a policy-based automation approach to make the definition of resources, and relationships of resources to each other, as easy as possible.

The Tivoli System Automation for Multiplatforms end-to-end component allows you to monitor and manage multiple clusters and relationships between them. For instance, there can be one cluster for the Tivoli Dynamic Workload Broker server, another for the Tivoli Workload Scheduler domain manager, and then a sysplex where the Tivoli Workload Scheduler z/OS end-to-end engine is running. Inter-cluster relations can be managed and enforced from the Tivoli System Automation end-to-end component.

Sample High Availability scenario

Figure 11-9 shows an integration example that meets requirements on high availability and short recovery times:

- The Tivoli Dynamic Workload Broker server is running on an AIX HACMP cluster.
- The Tivoli Workload Scheduler for z/OS engine is running in a z/OS sysplex. The standby engine feature is exploited, allowing automatic, semi-automatic, or manual take over from the active engine to the standby engine should an outage impact the active engine.
- The Tivoli Workload Scheduler for z/OS end-to-end server uses a Dynamic Virtual IP Address (DVIPA), allowing seamless connection switching to occur when the active engine moves to another z/OS image in the sysplex.

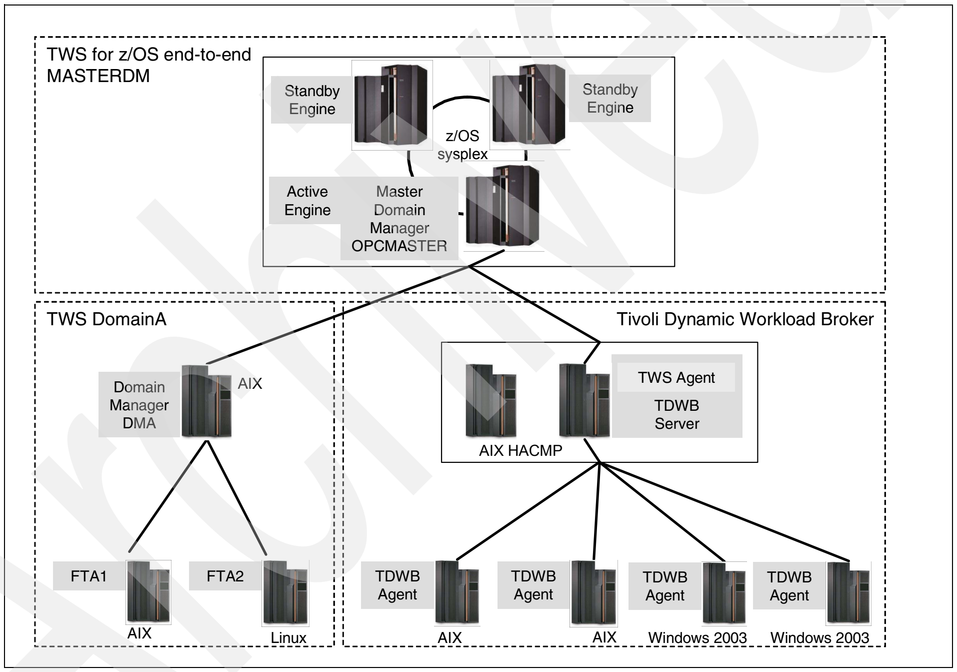


Figure 11-9 Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end high availability integration

Summary

In summary, several needs and requirements should be considered when selecting the architecture:

- The Tivoli Workload Scheduler for z/OS end-to-end domain topology.

  Tivoli Workload Scheduler for z/OS end-to-end does not limit the number of domains or levels (the hierarchy) in the network. There can be as many levels of domains as is appropriate for a given computing environment.

  The number of domains or levels should take into consideration the topology of the physical network where Tivoli Workload Scheduler for z/OS end-to-end and Tivoli Dynamic Workload Broker are installed. Most often, geographical boundaries are used to determine divisions between domains.

- Tivoli Workload Scheduler fault-tolerant agents and Tivoli Dynamic Workload Broker agents can coexist on the same distributed server. The Tivoli Workload Scheduler fault-tolerant agent is suitable for running housekeeping jobs on each individual server (for example, backups and log archiving).

- High availability and recovery requirements for the Tivoli Dynamic Workload Broker server (remember that the Tivoli Dynamic Workload Broker server and Tivoli Workload Scheduler agent run on the same physical machine).

  Tivoli Dynamic Workload Broker server availability and recovery requirements should be extended to include the Tivoli Workload Scheduler domain manager to which the Tivoli Workload Scheduler agent is connected. The reason for this is that the Tivoli Workload Scheduler agent can only launch jobs under the direction of its domain manager.

11.3 Installation and configuration considerations

Installation and configuration can be divided into two major parts. First, installation and configuration of the two products, and then performing the specific steps to integrate them.

11.3.1 Product installation and configuration

Before you can enable management of Tivoli Dynamic Workload Broker jobs using Tivoli Workload Scheduler for z/OS end-to-end you need to complete the following tasks:

- Install and configure IBM Tivoli Workload Scheduler for z/OS.
- Activate the end-to-end feature in the Tivoli Workload Scheduler for z/OS.
- Install and configure IBM Tivoli Dynamic Workload Broker server, agent manager, agents, and user interfaces.

Note: In this section we only focus on installation and customization activities and considerations that are specific when enabling management of Tivoli Dynamic Workload Broker jobs using Tivoli Workload Scheduler for z/OS end-to-end.

We assume that Tivoli Workload Scheduler for z/OS product installation tasks and Tivoli Workload Scheduler for z/OS end-to-end feature activation have been completed.

We also assume that the Tivoli Dynamic Workload Broker server, agent manager, agents, and user interfaces have been installed.

Product installation tasks are documented in:

- *IBM Tivoli Workload Scheduler for z/OS Installation Guide,* SC32-1264
- *IBM Tivoli Workload Scheduler for z/OS Customization and Tuning,* SC32-1265
- *IBM Tivoli Dynamic Workload Broker Installation and Configuration*

  Tivoli Dynamic Workload Broker installation is also the topic of Chapter 3, "Tivoli Dynamic Workload Broker installation" on page 81.

11.3.2 Overview of specific installation and customization steps

The specific steps that you need to perform are:

1. Plan your configuration.
2. Configure network connectivity.
3. Install and configure the Tivoli Workload Scheduler agent.
4. Create the Tivoli Workload Scheduler for z/OS end-to-end workstation to be used to connect the Tivoli Workload Scheduler agent.

5. Activate the Tivoli Workload Scheduler for the z/OS end-to-end workstation.
6. Verify the integration.

11.3.3 Plan your configuration

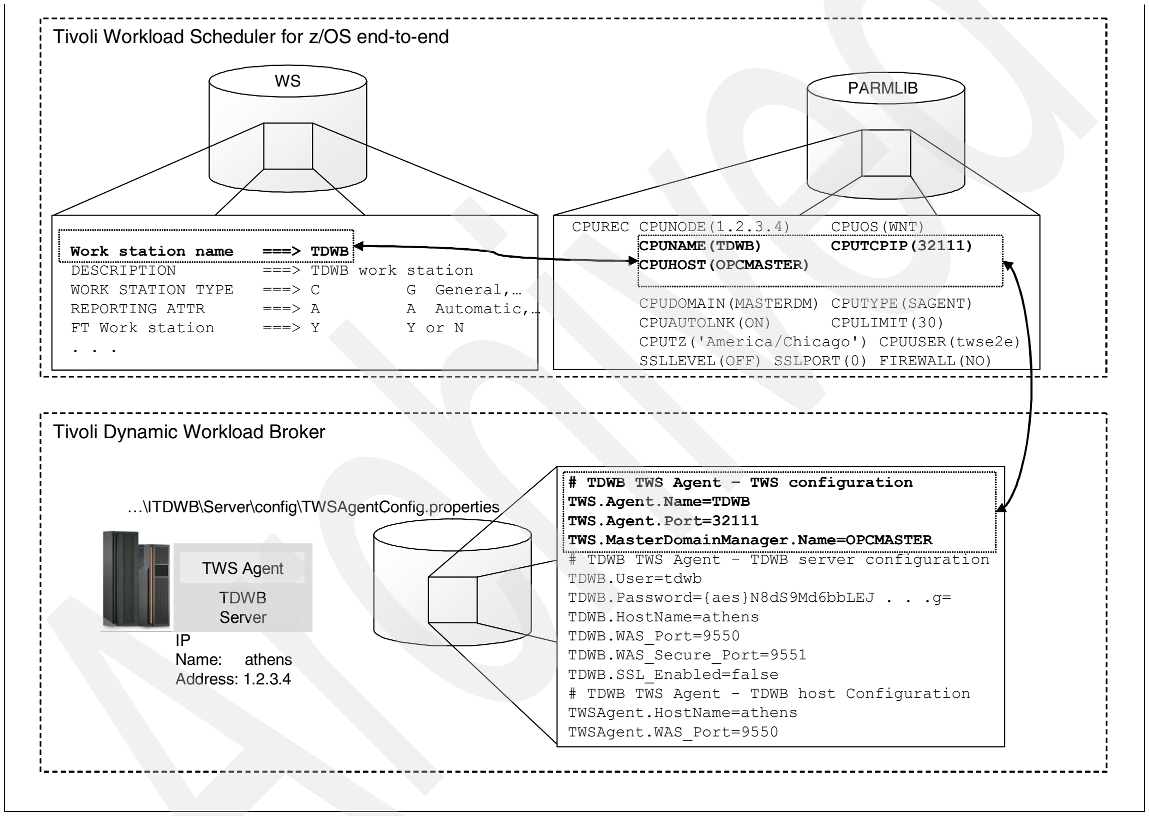
Planning you configuration involves:

1. Select the Tivoli Workload Scheduler for the z/OS end-to-end domain topology to use for the Tivoli Workload Scheduler agent. This includes identifying the domain manager that will host the Tivoli Workload Scheduler agent.
2. Choose a Tivoli Workload Scheduler for the z/OS end-to-end workstation name.
3. Identify the TCP/IP port to be used by the Tivoli Workload Scheduler agent.
4. Identify the TCP/IP DNS name and the IP address of the Tivoli Dynamic Workload Broker server.
5. Identify the TCP/IP DNS name and the IP address of the Tivoli Workload Scheduler for the z/OS server. If your Tivoli Workload Scheduler for the z/OS server is enabled to run on multiple z/OS systems, you need to identify DNS names and IP addresses of all z/OS systems involved.

   If you use Dynamic Virtual IP Addressing (DVIPA) for the Tivoli Workload Scheduler for z/OS server you also need to identify its DVIPA IP address.

Figure 11-10 shows how the Tivoli Workload Scheduler agent definition is related to Tivoli Workload Scheduler for z/OS end-to-end standard agent and Tivoli Workload Scheduler for z/OS workstation definitions.

> **Important:** The configuration shown in Figure 11-6 on page 533 and Figure 11-10 on page 542 is used as a working example for the installation and customization tasks described. You must change the workstation name, IP port, and so on to match your own configuration.


Tivoli Workload Scheduler for z/OS end-to-end
WS
PARMLIB
Tivoli Dynamic Workload Broker
…\ITDWB\Server\config\TWSAgentConfig.properties
TWS Agent
TDWB Server
IP
Name: athens
Address: 1.2.3.4

```
Work station name    ===> TDWB
DESCRIPTION          ===> TDWB work station
WORK STATION TYPE    ===> C         G  General,…
REPORTING ATTR       ===> A         A  Automatic,…
FT Work station      ===> Y         Y or N
. . .
```

```
CPUREC CPUNODE(1.2.3.4)     CPUOS(WNT)
       CPUNAME(TDWB)        CPUTCPIP(32111)
       CPUHOST(OPCMASTER)

       CPUDOMAIN(MASTERDM) CPUTYPE(SAGENT)
       CPUAUTOLNK(ON)      CPULIMIT(30)
       CPUTZ('America/Chicago') CPUUSER(twse2e)
       SSLLEVEL(OFF) SSLPORT(0) FIREWALL(NO)
```

```
# TDWB TWS Agent - TWS configuration
TWS.Agent.Name=TDWB
TWS.Agent.Port=32111
TWS.MasterDomainManager.Name=OPCMASTER
# TDWB TWS Agent - TDWB server configuration
TDWB.User=tdwb
TDWB.Password={aes}N8dS9Md6bbLEJ . . .g=
TDWB.HostName=athens
TDWB.WAS_Port=9550
TDWB.WAS_Secure_Port=9551
TDWB.SSL_Enabled=false
# TDWB TWS Agent - TDWB host Configuration
TWSAgent.HostName=athens
TWSAgent.WAS_Port=9550
```

Figure 11-10 Configuring parameters for the Tivoli Workload Scheduler agent

11.3.4 Configure network connectivity

The network must be configured to allow TCP/IP communication between the Tivoli Dynamic Workload Broker and the Tivoli Workload Scheduler for z/OS end-to-end.

On one end is Tivoli Workload Scheduler agent (which emulates a Tivoli Work Scheduler standard agent).

On the other end is the Tivoli Workload Scheduler for the z/OS end-to-end domain manager that hosts the Tivoli Workload Scheduler agent.

An example of the configuration definitions involved are shown in Example 11-1.

Example 11-1 Sample Tivoli Dynamic Workload Broker and TWS for z/OS TCP/IP related parameters

```
TWS for z/OS end-to-end USS server topology definition:
TOPOLOGY HOSTNAME(twsce2e)              /* TWSE2E USSserver hostname  */
         PORTNUMBER(32111)              /* TWSE2E netman listen port  */
         TCPIPJOBNAME(TCPIP)            /* z/OS TCP/IP started task   */
         ....
------------------------------------------------------------------------------------
TWS for z/OS end-to-end topology definition of TDWB TWS Agent workstation:
CPUREC   CPUNAME(TDWB)                   /* TWS end-to-end CPU name  */
         CPUNODE(1.2.3.4)                /* IP address or DNS name   */
         CPUTCPIP(32111)                 /* NETMAN TCP/IP port       */
         CPUHOST(OPCMASTER)              /* SAGENTs domain manager   */
         CPUTYPE(SAGENT)                 /* Agent type               */
         CPUDOMAIN(MASTERDM)             /* Domain name for WKST     */
         ...
====================================================================================
Tivoli Workload Scheduler agent TWSAgentConfig.properties file:
TWS.Agent.Name=TDWB
TWS.Agent.Port=32111
TWS.MasterDomainManager.Name=OPCMASTER
```

In Example 11-1 the Tivoli Workload Scheduler agent is directly connected to the master domain manager OPCMASTER.

Connectivity requirements

You must configure your TCP/IP network routing tables and fire walls to allow for the following connections to be established:

- The Tivoli Workload Scheduler agent is listening on the port specified in TWSAgentConfig. It listens for incoming requests from its hosting domain manager.
- The hosting domain manger is listening for incoming requests on a port specified in the TOPOLOGY PORTNUMBER parameter.
- The Tivoli Workload Scheduler agent will open connections:
  - *From* a random port on its own machine

  - *To* the domain manager listening port
- The domain manager will open connections
  - *From* a random port on its own machine
  - *To* the Tivoli Workload Scheduler agent listening port

Figure 11-11 shows the TCP/IP **netstat** command output from our lab environment (slightly edited to enhance readability).

```
netstat output on z/OS:
TWSCE2E       twsce2e:32111   0.0.0.0:0      Listening
TWSCE2E       twsce2e:32111   1.2.3.4:2893   Established
TWSCE2E       twsce2e:2584    1.2.3.4:32111  Established

netstat output on Windows server running TDWB server:
java.exe TCP   1.2.3.4:32111   twsce2e:2584   ESTABLISHED
java.exe TCP   1.2.3.4:2893    twsce2e:32111  ESTABLISHED
java.exe TCP   1.2.3.4:32111   0.0.0.0:0      LISTENING
```

Figure 11-11 TCP/IP netstat output

Important: You must configure the TCP/IP network connectivity using the DNS names, IP addresses, and IP ports for your own configuration.

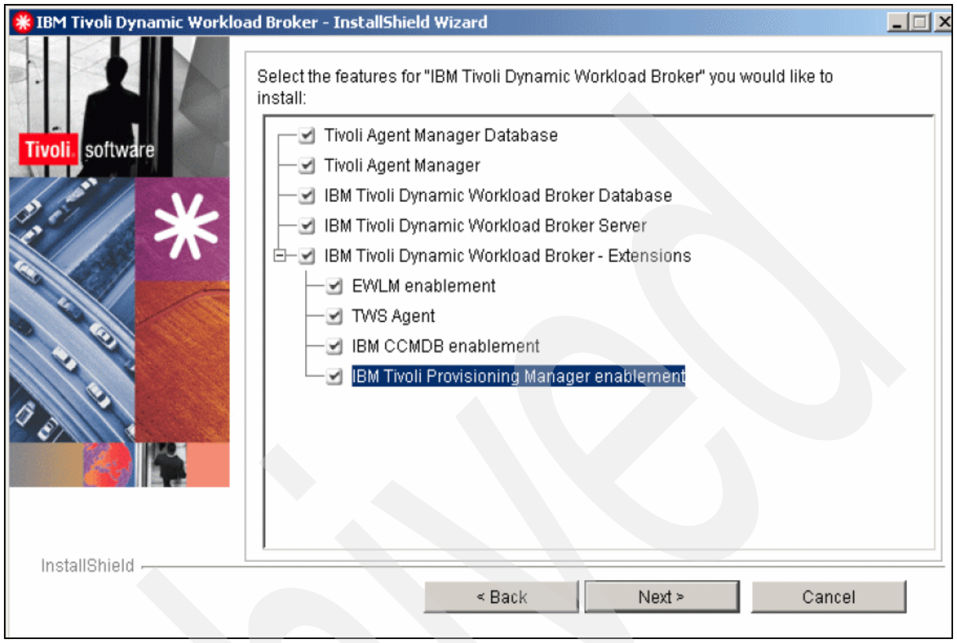
11.3.5 Installing and configuring the Tivoli Workload Scheduler agent

The Tivoli Workload Scheduler agent emulates the behavior of a Tivoli Workload Scheduler standard agent and is installed as an extension to the Tivoli Dynamic Workload Broker server.

Choosing the installation method

You install the Tivoli Dynamic Workload Broker server, including the Tivoli Workload Scheduler agent, by performing a custom installation using either the installation wizard or the silent installation method.

When performing a custom installation using the installation wizard method, the features window is displayed (Figure 11-12).


IBM Tivoli Dynamic Workload Broker - InstallShield Wizard

Select the features for "IBM Tivoli Dynamic Workload Broker" you would like to install:

- ✓ Tivoli Agent Manager Database
- ✓ Tivoli Agent Manager
- ✓ IBM Tivoli Dynamic Workload Broker Database
- ✓ IBM Tivoli Dynamic Workload Broker Server
- ✓ IBM Tivoli Dynamic Workload Broker - Extensions
  - ✓ EWLM enablement
  - ✓ TWS Agent
  - ✓ IBM CCMDB enablement
  - ✓ IBM Tivoli Provisioning Manager enablement

Figure 11-12 Installation wizard features window - TWS Agent selected

Important: The Tivoli Workload Scheduler agent will not be installed if you perform a typical installation of the Tivoli Dynamic Workload Broker server.

You can only install one Tivoli Workload Scheduler agent.

Tivoli Workload Scheduler agent configuration

To configure the Tivoli Workload Scheduler agent for integration with Tivoli Workload Scheduler for z/OS end-to-end you need to supply the following information during installation:

- TWS workstation name

  Specify the name of the Tivoli Workload Scheduler for z/OS workstation that represents the Tivoli Workload Scheduler agent.

- Tivoli Workload Scheduler Domain Manager name

  If the Tivoli Workload Scheduler agent is directly connected to the Tivoli Workload Scheduler for the z/OS end-to-end server, this name must be set to OPCMASTER.

  Otherwise, you specify the workstation name of the Tivoli Workload Scheduler for the z/OS end-to-end domain manager that is hosting the Tivoli Workload Scheduler agent.

- TWS Agent Port

  Specify the TCP/IP port number to be used by the Tivoli Workload Scheduler agent. It consists of five numerals and, if omitted, uses the default value, 31111. The Tivoli Workload Scheduler agent uses this port to listen for incoming requests from its domain manager.

- ITDWB user name

  The user account name is used by the Tivoli Workload Scheduler agent when authenticating itself to the Tivoli Dynamic Workload Broker server.

- ITDWB user password

  The user account password.

Note on Tivoli Workload Scheduler for z/OS workstation names:

A Tivoli Workload Scheduler for z/OS end-to-end workstation name can have a maximum of four characters, must start with a letter, and all letters must be uppercase. There is one exception — the master domain manager has the fixed name OPCMASTER.

Note on ITDWB user name and password:

If WebSphere security is enabled on the Tivoli Dynamic Workload Broker server, the Tivoli Workload Scheduler agent must authenticate (to the server) while submitting and managing job requests coming from the Tivoli Workload Scheduler for the z/OS end-to-end environment.

TWSAgentConfig.properties file

The Tivoli Workload Scheduler agent configuration information is stored in the TWSAgentConfig.properties file, which is located in the Tivoli Dynamic Workload Broker installation subdirectory config (<*installation directory*>\config).

Example 11-2 shows a sample TWSAgentConfig.properties file.

Example 11-2 Sample TWSAgentConfig.properties file

```
# ---------------------------------
# TWS Standard Agent Configuration
# ---------------------------------
TWS.Agent.Name=TDWB
TWS.Agent.Port=32111
TWS.MasterDomainManager.Name=OPCMASTER
#--------------------------
# TDWB Server Configuration
#--------------------------
TDWB.User=tdwb
TDWB.Password={aes}N8dS9Md6bbLEJntpEi3Z1GsJC4E8UD+BihdSYuSjffg=
TDWB.HostName=athens
TDWB.WAS_Port=9550
TDWB.WAS_Secure_Port=9551
TDWB.SSL_Enabled=false
#-----------------------------
# TWS Agent Host Configuration
#-----------------------------
TWSAgent.HostName=athens
TWSAgent.WAS_Port=9550
```

Making changes to the configuration

If you need to modify the Tivoli Workload Scheduler agent configuration, update the TWSAgentConfig.properties file. To activate the changes you must restart the Tivoli Dynamic Workload Broker server.

Verifying installation of the Tivoli Workload Scheduler agent

After you have completed installation and restarted the Tivoli Dynamic Workload Broker server, the Tivoli Workload Scheduler agent starts. The Tivoli Workload Scheduler agents starts automatically during server startup. Example 11-3 shows sample messages confirming that the Tivoli Workload Scheduler agent has started.

Example 11-3 Tivoli Workload Scheduler agent startup messages in TDWB server SystemOut.log

```
[3/28/07 3:21:22:547 PST] 0000000a TWSAgent      I
AWKTSA005I The TWS Agent has been successfully configured.
[3/28/07 3:21:22:594 PST] 0000000a TWSAgent      I
AWKTSA001I The TWS Agent has been successfully started.
[3/28/07 3:21:22:594 PST] 00000034 TWSAgent      I
AWKTSA007I Netman has been successfully started and it is listening on port 32111.
```

11.3.6 Create Tivoli Workload Scheduler for z/OS end-to-end

Creating a Tivoli Workload Scheduler for z/OS end-to-end workstation consists of two steps:

1. Create a Tivoli Workload Scheduler for z/OS workstation.
2. Create the topology definition for the workstation.

Create a Tivoli Workload Scheduler for z/OS workstation

Create a Tivoli Workload Scheduler for z/OS workstation using the Tivoli Workload Scheduler for z/OS ISPF panels. Enter information like that shown in Figure 11-13. Make sure that you define it as a computer type workstation with reporting attribute automatic and FT workstation flag set to yes.

```
workstation name      : TDWB
DESCRIPTION          ===> TDWB ITSO Austin on Windows 2003
workstation TYPE    ===> C        G  General, C  Computer, P  Printer
REPORTING ATTR       ===> A         A  Automatic, S  Manual start and completion
                                    C  Completion only, N  Non reporting
FT workstation      ===> Y        FT workstation, Y or N
PRINTOUT ROUTING     ===> SYSPRINT  The ddname of daily plan printout data set
SERVER USAGE         ===> N         Parallel server usage C , P , B or N
DESTINATION          ===> ________  Name of destination
Options:  allowed  Y or N
 SPLITTABLE          ===> N         JOB SETUP         ===> N
 STARTED TASK, STC   ===> N         WTO               ===> N
 AUTOMATION          ===> N         WAIT              ===> N
Defaults:
 TRANSPORT TIME      ===> 00.00     Time from previous workstation HH.MM
 DURATION            ===> ________  Duration for a normal operation HH.MM.SS
Last updated by        : CCBJW    on 07/03/13 at 19.37
```

Figure 11-13 Sample Tivoli Workload Scheduler for z/OS V8.3 workstation definition

Create the topology definition for the workstation

Create the topology definition by editing the topology definition member used by the Tivoli Workload Scheduler for z/OS end-to-end. Add a CPUREC definition like in Example 11-4.

Example 11-4 Tivoli Workload Scheduler for z/OS V8.3 end-to-end workstation topology definition

```
CPUREC   CPUNAME(TDWB)                /* TWS end-to-end CPU name  */
         CPUOS(WNT)                   /* Operating system         */
         CPUNODE(1.2.3.4)             /* IP address or DNS name   */
         CPUTCPIP(32111)              /* NETMAN TCP/IP port       */
         CPUDOMAIN(MASTERDM)          /* Domain name for WKST     */
         CPUTYPE(SAGENT)              /* Agent type               */
         CPUHOST(OPCMASTER)           /* SAGENT domain manager    */
         CPUAUTOLNK(ON)               /* Automatic link ON/OFF    */
         CPULIMIT(30)                 /* Max. # parallel jobs     */
         CPUTZ('America/Chicago')     /* Agent time zone          */
         SSLLEVEL(OFF)                /* SSL auth. between SA-DM? */
         SSLPORT(0)                   /* SSL auth is not required */
         FIREWALL(NO)                 /* FW between SA and DM?    */
```

Important: CPUNAME(), CPUTCPIP() and CPUHOST() parameters must match the configuration of the Tivoli Workload Scheduler agent. See Example 11-2 on page 547.

Tip: The CPULIMIT() parameter specifies the number of jobs (0 to 1024) that can run at the same time in the workstation. If you specify 0, no jobs are launched on the workstation. For fault-tolerant and standard agents having a direct connection to the end-to-end server, changes in this value come into effect during the extension or replanning of the current plan, not during the renewal of the symphony file.

Restriction: Tivoli Dynamic Workload Broker V1.1 does not support SSL authentication between the domain manager and the Tivoli Workload Scheduler agent. Specify SSLLEVEL(OFF).

Note: The CPUOS() parameter is not used by the Tivoli Dynamic Workload Broker V1.1 Tivoli Workload Scheduler agent. It can be set to either CPUOS(WNT) or CPUOS(AIX).

11.3.7 Activate the Tivoli Workload Scheduler for z/OS end-to-end workstation

To activate the new Tivoli Workload Scheduler for z/OS end-to-end workstation you must update the current plan using either the current plan extend or the current plan replan batch job.

The current plan extend/replan job will as part of its processing:

- Add the new workstation to the current plan.
- Add the new standard agent workstation to the symphony file. Topology information for the workstation will also be added to the symphony file.

When the symphony file is ready, the Tivoli Workload Scheduler for z/OS engine will send it to the distributed network of domain managers and fault-tolerant agents and end-to-end scheduling will continue.

Tip: Consider specifying CPUAUTOLNK(ON) in the topology definition so that when the domain manager that hosts the Tivoli Workload Scheduler agent starts, it automatically links the workstation. If you specify CPUAUTOLNK(OFF) you must submit a link command manually from the Modify Current Plan dialog or Job Scheduliing Console.

Verify activation

You can verify that the activation was successful by checking that the workstation is available in the current plan and that its status is linked and active. If you specified CPUAUTOLNK(OFF) you must first submit a link command manually.

You can also verify activation by checking messages in the Tivoli Workload Scheduler for z/OS logs and the Tivoli Dynamic Workload Broker server log.

Example 11-5 shows messages from the Tivoli Workload Scheduler for z/OS Controller log.

Example 11-5 TWS for z/OS Controller workstation activation messages

```
EQQWL10W workstation TDWB HAS BEEN SET TO LINKED STATUS TYPE SAGENT DOMAIN MASTERDM
EQQWL10W workstation TDWB HAS BEEN SET TO ACTIVE STATUS TYPE SAGENT DOMAIN MASTERDM
```

Example 11-6 shows messages from Tivoli Workload Scheduler for z/OS end-to-end server TWSMERGE log.

Example 11-6 TWS for z/OS end-to-end workstation activation messages

```
MAILMAN:AWSBCV087I Added TDWB to workstation table, node "R", server " ", link type
"1", flags "273".
BATCHMAN:AWSBHT030I Adding workstation TDWB to be scheduled by host workstation
OPCMASTER.
BATCHMAN:AWSBDY103I Received command MY:UNLINK for run number 11 for workstation TDWB
from workstation OPCMASTER.
BATCHMAN:AWSBHT034I Host workstation OPCMASTER sent an INIT record to workstation
TDWB.
MAILMAN:AWSBCV029I Attempting to link to TDWB.
MAILMAN:AWSBCV061I Starting to initialize TDWB
TDWB:WRITER:AWSBCW028I Started by MAILMAN/8.3 from TDWB; workstation type: WNT
TDWB:WRITER:AWSBCW031I Handshake command_type StartMailbox
MAILMAN:AWSBCV055I Finished initializing TDWB
MAILMAN:AWSBCT041I Service 2002 started on TDWB
MAILMAN:AWSBCV104I Has linked to TDWB using TCP.
BATCHMAN:AWSBDY104I Received command MY:LINK for run number 12 for workstation TDWB
from workstation OPCMASTER.
BATCHMAN:Workstation TDWB State is being changed: LINKED=TCP
BATCHMAN:AWSBDY112I Received command MY:WRITER-UP for run number -1 for workstation
TDWB from workstation OPCMASTER.
BATCHMAN:Workstation TDWB State is being changed: WRITER
BATCHMAN:AWSBDY110I Received command MY:JOBMAN-UP for run number 12 from workstation
TDWB.
BATCHMAN:Workstation TDWB State is being changed: JOBMAN
BATCHMAN:AWSBDY105I Received command MY:UNSET INIT for run number 12 from workstation
TDWB.
BATCHMAN:Workstation TDWB State is being changed: UNSETTING: INITTED
BATCHMAN:AWSBHT035I Workstation TDWB completed its INIT process. 0 jobs running.
BATCHMAN:AWSBDY106I Received command MY:INIT for run number 12 from workstation TDWB.
BATCHMAN:Workstation TDWB State is being changed: INITTED
BATCHMAN:AWSBHT033I Workstation TDWB is now active, scheduling is resuming.
BATCHMAN:AWSBHT035I Workstation TDWB completed its INIT process. 1 jobs running.
BATCHMAN:AWSBDY105I Received command MY:UNSET INIT for run number 12 from workstation
TDWB.
BATCHMAN:AWSBDY106I Received command MY:INIT for run number 12 from workstation TDWB.
```

Example 11-7 show messages from the Tivoli Dynamic Workload Broker server SystemOut.log file.

Example 11-7 Tivoli Workload Scheduler agent activation messages in TDWB server SystemOut.log

```
[3/28/07 4:13:55:469 PST] 00000034 TWS Agent...
...AWKTSA014I The STOP MAILMAN service request has been received by Netman.
...AWKTSA010I The START WRITER service request has been received by Netman.
...AWKTSA017I Writer has been successfully started and it is listening for messages.
...AWKTSA032I The TWS Agent CPU has been successfully linked to
[Ljava.lang.Object;@466a6273 using port {1}.
...AWKTSA026I Jobman has been successfully started.
...AWKTSA021I Mailman has been successfully started and the uplink connection is
established.
...AWKTSA012I The START MAILMAN service request has been received by Netman.
...AWKTSA013W The MAILMAN service is already started.
...AWKTSA034I The TWS Agent CPU has been successfully initialized.
```

11.3.8 Verify integration

The final verification of the Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end integration is to schedule a job end-to-end. To do this you must:

- Create a Tivoli Workload Scheduler for z/OS end-to-end job stream.
- Create a Tivoli Workload Scheduler for z/OS end-to-end job SCRPTLIB member.
- Create a Tivoli Dynamic Workload Broker Job definition.

11.4 Planning and choreography

Planning and choreography in an integrated Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end environment means that you map business applications and tasks to what in Tivoli Workload Scheduler for z/OS end-to-end terminology are units of work called jobs. Jobs are grouped into job streams along with times, priorities, and other dependencies that determine the exact order of the jobs. Job streams also contain business-run schedules (daily, weekly, cyclic every hour, and so on).

In this section we focus on activities and considerations in the following planning and choreography areas:

- Integration benefits
- Allocation of jobs to computer resources
- Job definition user interfaces
- Defining job and job stream definitions
  - Job and job stream definitions in Tivoli Workload Scheduler for z/OS end-to-end
  - Job definitions in Tivoli Dynamic Workload Broker
  - Interrelation of job definitions
- Moving existing jobs between the environments

Note: In Tivoli Workload Scheduler for z/OS end-to-end the terms *job stream* and *application description* (AD) are synonyms. In this section we have chosen only to use the term job stream.

11.4.1 Integration benefits

Integrating the Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end delivers planning and choreography benefits both ways.

The Tivoli Dynamic Workload Broker provides a series of improvements on your existing IBM Tivoli Workload Scheduler for the z/OS end-to-end solution:

- Virtualization of the scheduling infrastructure by providing an abstraction layer on the resource selection
- Workload balancing by routing jobs among a group of resources according to the availability and activity levels of those resources
- SOA job brokering services
- Scheduling of IBM WebSphere Java 2 Enterprise Edition (J2EE) applications
- Automatic routing of jobs to the most appropriate resources based on job requirements
- Enhanced flexibility in workload distribution and running
- Automatic routing of jobs for which submission failed to appropriate resources

Tivoli Workload Scheduler for z/OS end-to-end provides the following key features to the Tivoli Dynamic Workload Broker:

- End-to-end scheduling infrastructure
- Advanced scheduling, calendaring, planning, and choreographing capabilities
- Job restart and recovery capabilities

11.4.2 Allocation of jobs to computer resources

Allocation of jobs to computer resources can be done in either of two ways:

- Using Tivoli Workload Scheduler fault tolerant agents

  A job can be statically allocated to a dedicated computer resource. This is done using a Tivoli Workload Scheduler for z/OS end-to-end job definition that specifies that the job will run on a traditional Tivoli Workload Scheduler fault tolerant workstation.

- Using Tivoli Dynamic Workload Broker

  A job can be dynamically allocated to computer resources. This is done using two complementary job definitions:

  – A Tivoli Workload Scheduler for z/OS end-to-end job definition that specifies that the job will run on the workstation representing Tivoli Dynamic Workload Broker.

  – A Tivoli Dynamic Workload Broker job definition. This job definition defines how the job should be allocated to physical or virtual resources according to the job importance, requirements, and scheduling policies.

Which type of allocation to choose is often driven by business needs and requirements in areas such as job SLA and computer resource optimization.

The following job types might be potential candidates for Tivoli Dynamic Workload Broker jobs:

- Heavy computational tasks
- Jobs that require resource optimization and resource balancing
- Jobs strongly dependent on the availability of hardware resources, such as CPU and RAM
- Jobs that must run in high-availability clusters
- Jobs that can benefit from automated recovery of failures
- Jobs requiring IBM WebSphere job scheduling

11.4.3 Job definition user interfaces

In an integrated environment you need two complementary job definitions for every job that is going to be submitted by Tivoli Workload Scheduler for z/OS end-to-end to Tivoli Dynamic Workload Broker.

You interact with Tivoli Workload Scheduler for z/OS end-to-end user interfaces (see also “User interfaces” on page 521) to define the Tivoli Workload Scheduler for z/OS end-to-end job stream and job definition parts:

Job stream	The job stream is used to define job details, job dependencies, and job stream run cycle information
SCRPTLIB job member	The SCRPTLIB job member is used to define the name of the Tivoli Dynamic Workload Broker job to be run. The member acts as the link between the job definition in the job stream and the Tivoli Dynamic Workload Broker job definition (located in the Tivoli Dynamic Workload Broker job repository).

Job stream definitions are maintained using Job Scheduling Console or Tivoli Workload Scheduler for z/OS ISPF panels.

SCRPTLIB job members are maintained using z/OS ISPF panels.

You interact with Dynamic Workload Broker user interfaces to define the Dynamic Workload Broker job definition part. You use the Tivoli Dynamic Workload Broker Web Console or the Job Brokering Definition Console. Refer to Chapter 4, “Working with Tivoli Dynamic Workload Broker” on page 141, for more details on how to use these interfaces.

The complementary job definitions are stored in different physical places and access to them is protected by different mechanisms. Each person with a need to maintain the definitions needs one or more of the following types of user credentials:

- ► Tivoli Workload Scheduler for z/OS end-to-end access
  - – A z/OS TSO user ID is required to use the Job Scheduling Console, Tivoli Workload Scheduler for z/OS ISPF panels, and z/OS ISPF.

    Authorization to Tivoli Workload Scheduler for z/OS data and to the SCRPTLIB job member is verified using this user ID.
  - – A user name is required to use the Job Scheduling Console. (This user name is mapped to a z/OS TSO user ID when connecting to the Tivoli Workload Scheduler for z/OS engine.)

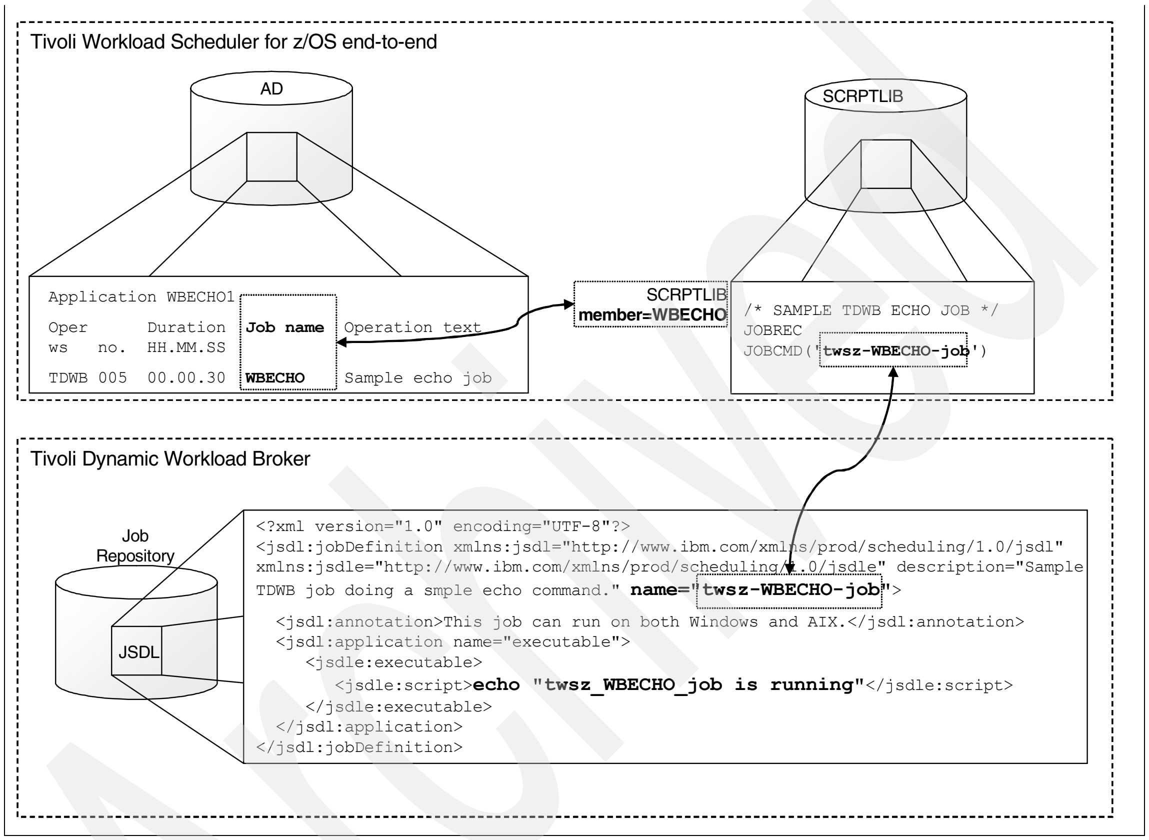
- Tivoli Dynamic Workload Broker access
  - A user ID is required to use the Tivoli Dynamic Workload Broker Web Console.

    An additional user name is most likely required for the Web Console server connection. This user name is used for authentication to the Tivoli Dynamic Workload Broker server. Authorization to Tivoli Dynamic Workload Broker data is verified using this user name.
  - A user name is required in the Job Brokering Definition Console (JBDC). The user name is needed in a server connection used by JBDC when uploading and downloading job definitions to and from the Tivoli Dynamic Workload Broker server.

Refer to Chapter 11, “Managing Tivoli Dynamic Workload Broker jobs using Tivoli Workload Scheduler for z/OS end-to-end” on page 515, for more information about authentication mechanisms in the Tivoli Dynamic Workload Broker.

11.4.4 Defining job and job stream definitions

Figure 11-14 shows how a Tivoli Workload Scheduler for z/OS end-to-end job stream and job definition are related to the Tivoli Dynamic Workload Broker job definition.



Tivoli Workload Scheduler for z/OS end-to-end
AD
SCRPTLIB
SCRPTLIB
member=WBECHO
Tivoli Dynamic Workload Broker
Job Repository
JSDL

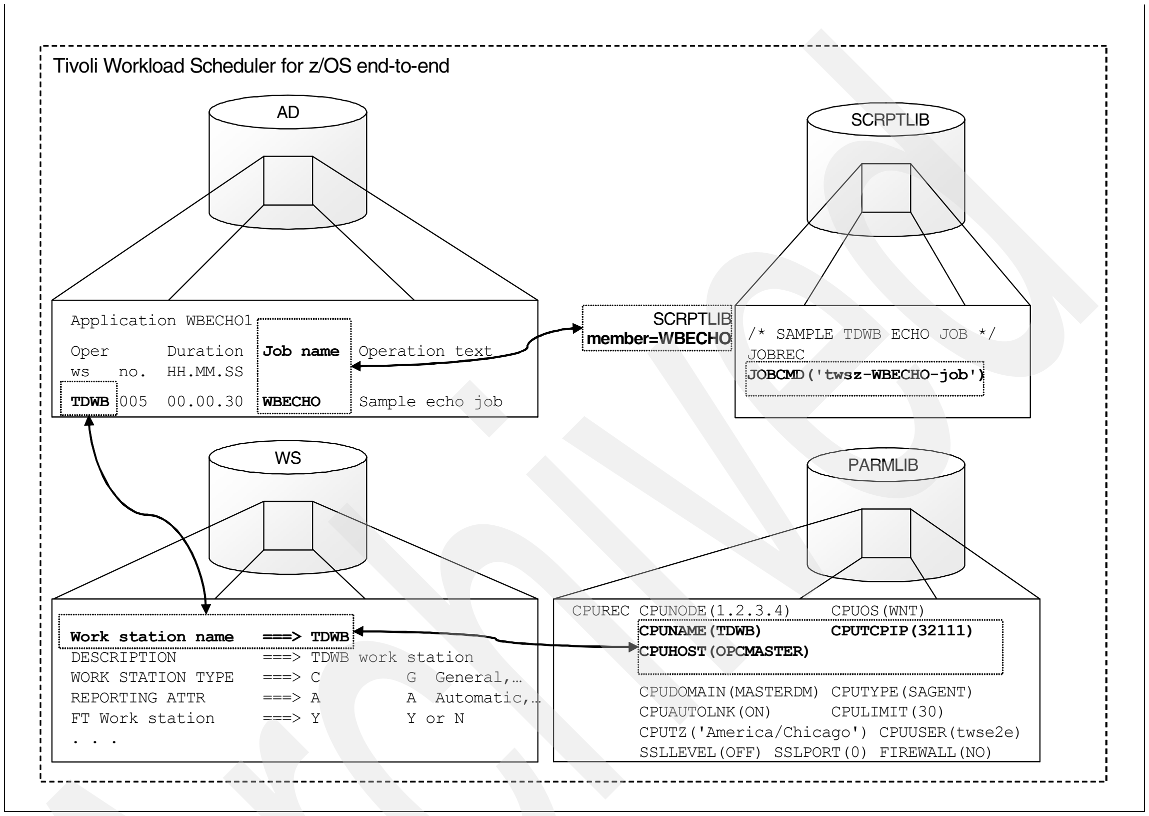
```
Application WBECHO1

Oper       Duration   Job name    Operation text
ws    no.  HH.MM.SS

TDWB 005   00.00.30   WBECHO      Sample echo job
```

```
/* SAMPLE TDWB ECHO JOB */
JOBREC
JOBCMD('twsz-WBECHO-job')
```

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" description="Sample
TDWB job doing a smple echo command." name="twsz-WBECHO-job">
  <jsdl:annotation>This job can run on both Windows and AIX.</jsdl:annotation>
  <jsdl:application name="executable">
     <jsdle:executable>
        <jsdle:script>echo "twsz_WBECHO_job is running"</jsdle:script>
     </jsdle:executable>
  </jsdl:application>
</jsdl:jobDefinition>
```

Figure 11-14 Interrelation of TWS for z/OS end-to-end job stream and TDWB job definition

You define that a job is to be submitted to the Tivoli Dynamic Workload Broker by assigning it to the workstation that is representing the Tivoli Workload Scheduler agent. Figure 11-15 shows how the job stream and job definition are related to the workstation definition.



```
Application WBECHO1
Oper     Duration   Job name  Operation text
ws   no. HH.MM.SS
TDWB 005 00.00.30   WBECHO    Sample echo job
```

```
/* SAMPLE TDWB ECHO JOB */
JOBREC
JOBCMD('twsz-WBECHO-job')
```

```
Work station name    ===> TDWB
DESCRIPTION          ===> TDWB work station
WORK STATION TYPE    ===> C        G  General,…
REPORTING ATTR       ===> A        A  Automatic,…
FT Work station      ===> Y        Y or N
. . .
```

```
CPUREC CPUNODE(1.2.3.4)      CPUOS(WNT)
       CPUNAME(TDWB)         CPUTCPIP(32111)
       CPUHOST(OPCMASTER)

       CPUDOMAIN(MASTERDM)   CPUTYPE(SAGENT)
       CPUAUTOLNK(ON)        CPULIMIT(30)
       CPUTZ('America/Chicago') CPUUSER(twse2e)
       SSLLEVEL(OFF) SSLPORT(0) FIREWALL(NO)
```

Figure 11-15 Interrelation of TWS for z/OS end-to-end job stream and workstation definition

Tivoli Workload Scheduler for z/OS end-to-end job stream and job definitions are defined and maintained almost like any other distributed job. However, you need to be aware of the following considerations when defining a Tivoli Dynamic Workload Broker job:

- Non-centralized script versus centralized script type job

  The Tivoli Dynamic Workload Broker V1.1 Tivoli Workload Scheduler agent (which emulates a Tivoli Workload Scheduler standard agent) only supports *non-centralized* script type jobs. It cannot handle centralized script type jobs.

- SCRPTLIB JOBREC JOBCMD() and JOBSCR()() parameters

  The JOBCMD() or the JOBSCR() parameter must contain the name of the Tivoli Dynamic Workload Broker job definition to be run. The name is case sensitive. Make sure that your editor on z/OS does not translate to upper case when editing SCRPTLIB members.

  You can use either JOBCMD() or JOBSCR() to specify the name of the Tivoli Dynamic Workload Broker job definition to be run. It makes no difference which one you use.

- SCRPTLIB JOBREC JOBUSR() parameter

  Tivoli Dynamic Workload Broker V1.1 does not support the JOBUSR() parameter.

  You can define user impersonation credentials in the TDWB job definition. However, Tivoli Dynamic Workload Broker V1.1 only supports it for jobs running on UNIX and Linux.

  Tivoli Dynamic Workload Broker jobs running on Windows execute under the same user account as the Tivoli Dynamic Workload Broker agent.

  Tivoli Dynamic Workload Broker jobs running on UNIX and Linux execute under the same user account as the Tivoli Dynamic Workload Broker agent unless user impersonation credentials are defined in the TDWB job definition.

- SCRPTLIB JOBREC INTRACTV(YES) parameter

  Tivoli Dynamic Workload Broker V1.1 does not support the INTRACTV(YES) parameter.

- SCRPTLIB JOBREC RCONDSUC() parameter

  Tivoli Dynamic Workload Broker V1.1 does not support the RCONDSUC() parameter.

Tip: The *external jobname* (that is, the *operation extended name*) in the Tivoli Workload Scheduler for z/OS end-to-end job definition can be used to contain the Tivoli Dynamic Workload Broker job name specified in the JOBREC JOBCMD() parameter. By doing this you enable the possibility to search for jobs in the current plan using the Tivoli Dynamic Workload Broker job name.

Tivoli Dynamic Workload Broker job definitions require no special considerations before they can be used when building job streams and jobs in Tivoli Workload Scheduler for z/OS end-to-end. Refer to Chapter 11, “Managing Tivoli Dynamic Workload Broker jobs using Tivoli Workload Scheduler for z/OS end-to-end” on page 515, for details on how to define and maintain the Tivoli Dynamic Workload Broker job definitions.

JOBREC JOBCMD() and JOBSCR() parameter considerations

Specifying the JOBCMD() or JOBSCR() parameter in SCRPTLIB members requires some special considerations:

- The parameter must begin with the Tivoli Dynamic Workload Broker job name. The name is followed by workload broker specific keywords as appropriate. The generic format is:

  ```
  JOBCMD('<jobName> [-var <varName>=<varValue>,...] [-affinity ...]')
  ```

- Keep in mind that Tivoli Dynamic Workload Broker entity names are case sensitive.
- If the parameter includes more than one word, it must be enclosed within single or double quotation marks.

Example 11-8 shows examples of JOBCMD() usage.

Example 11-8 Specifying SCRPTLIB JOBREC JOBCMD() parameter

```
JOBCMD('twsz-WBECHO-job')
JOBCMD('twsz-tdwbcli-cmd -var tdwbcmdparms=-alias "TDWB#&OADID*"')
JOBCMD('twsz-WBECHO-job -twsAffinity jobname=J005_WBAFJ1')
```

11.4.5 Moving existing jobs between the environments

You can move an existing Tivoli Dynamic Workload Broker job to be managed and scheduled by the Tivoli Workload Scheduler for z/OS end-to-end. You can also move an existing Tivoli Workload Scheduler for z/OS end-to-end job to be submitted by the Tivoli Dynamic Workload Broker.

Moving an existing Tivoli Dynamic Workload Broker job

To move an existing Tivoli Dynamic Workload Broker job to be managed and scheduled by the Tivoli Workload Scheduler for z/OS end-to-end you need to perform the following steps:

1. Collect Tivoli Dynamic Workload Broker job information.
   a. Identify the Tivoli Dynamic Workload Broker job brokering name. Keep in mind that the name is case sensitive.
   b. Identify any job definition variables that need to be supplied at run time.
   c. Identify estimated job execution time.
   d. Identify run schedules needed (ad hoc, daily, and so on). Also identify whether the job must not be submitted before a specific time.

2. Plan and choreograph Tivoli Workload Scheduler for z/OS end-to-end.
   a. Identify the Tivoli Workload Scheduler for z/OS end-to-end workstation to be used for the Tivoli Dynamic Workload Broker job.
   b. Select an existing job stream or create a new job stream.
      i. Specify a suitable job stream name.
      ii. Add or modify the job stream run cycle as appropriate.
      iii. Add a new job to the selected job stream. Specify a suitable TWS job name, duration time, and operation number, and also specify the workstation. (Keep in mind that the TWS job name is not the same as the Tivoli Dynamic Workload Broker job name.)
      iv. Define the new job as a non-centralized type job.
      v. Optionally, add a job description.
      vi. Add internal and external job dependencies as appropriate.
      vii. Add job time information to make job time dependent as appropriate.
      viii. Save the job stream.
   c. Create a SCRPTLIB member.
      i. Create a SCRIPTLIB member with a name that equals the TWS job name specified in the job stream.
      ii. Add a JOBREC keyword.
      iii. Add a JOBCMD('<jobName>') parameter. Keep in mind that Tivoli Dynamic Workload Broker job brokering names are case sensitive. Also add Tivoli Dynamic Workload Broker job definition variables and values as appropriate.
      iv. Save the SCRPTLIB member.

Moving an existing Tivoli Workload Scheduler for z/OS job

Here our task is to move an existing Tivoli Workload Scheduler for z/OS end-to-end job to be submitted by Tivoli Dynamic Workload Broker. Let us begin by looking at a before and after picture of the job submission process.

Figure 11-16 shows how job submission works for a non-centralized type job J1 in Tivoli Workload Scheduler for z/OS end-to-end. When job J1 is ready to run the fault-tolerant agent (AIX1) retrieves the job script definition from the local symphony file (1) and submits the job to computer AIX-1 (2).

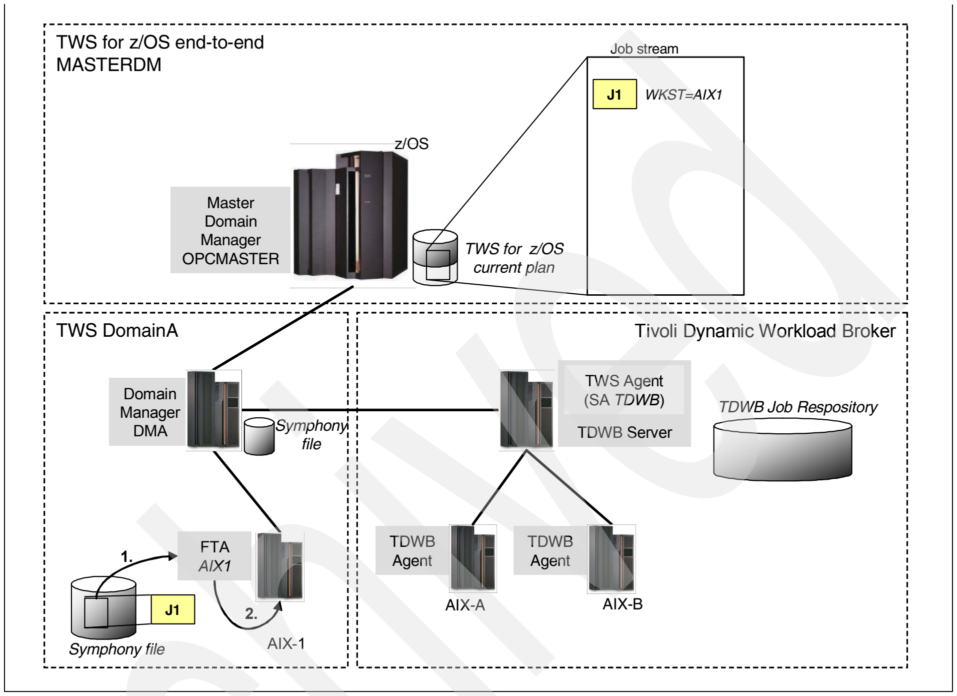


Figure 11-16 Job submission by TWS for z/OS end-to-end fault tolerant agent

Note: There is also another case where the job is a centralized script type job. When a centralized script type job is ready to run, the Tivoli Workload Scheduler for z/OS end-to-end engine retrieves the corresponding script member and sends it to the fault-tolerant agent (FTA). The FTA submits the job to the operating system. This submission process is not shown in Figure 11-16 on page 562.

Figure 11-17 shows how job submission works for a Tivoli Workload Scheduler for z/OS end-to-end job J1N running on a Tivoli Dynamic Workload Broker workstation TWDB.

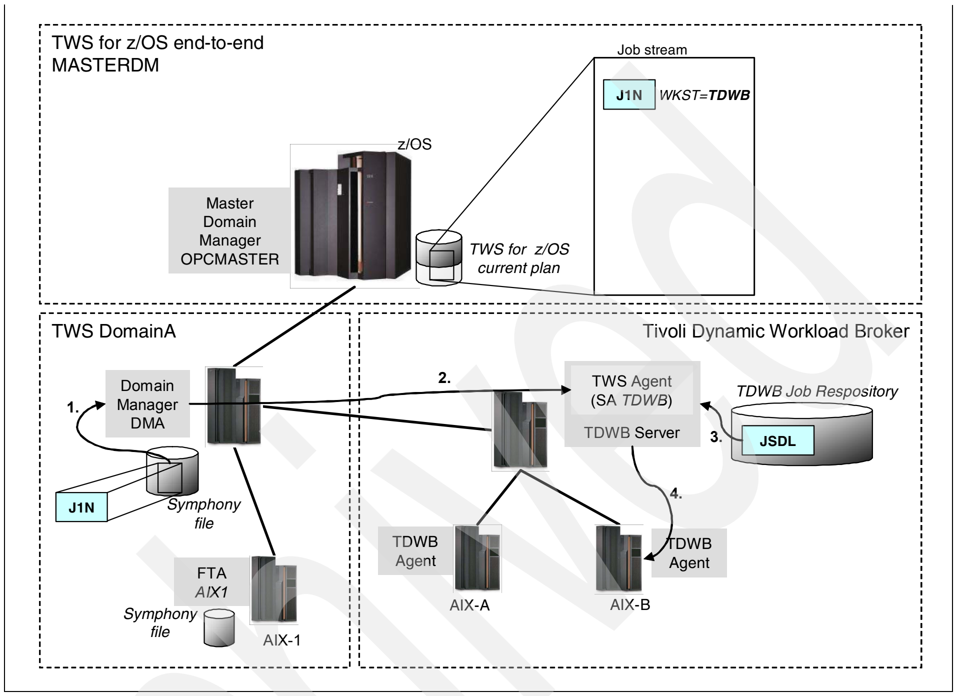


Figure 11-17 Job submission by Tivoli Dynamic Workload Broker

In the job submission scenario in Figure 11-17, Tivoli Workload Scheduler for z/OS end-to-end is only responsible for initiating a job start request. When job J1N is ready to run:

1. Domain manager DMA retrieves the job definition from the its local symphony file.
2. Domain manager DMA submits a job start request to the Tivoli Workload Scheduler agent, which is running inside the Tivoli Dynamic Workload Broker.
3. The Tivoli Dynamic Workload Broker server retrieves the JSDL job definition requested. It analyzes the job's resource requirements and according to the availability and activity levels of those resources it dynamically assigns a resource to the job. In this case resource AIX-B is assigned.
4. The Tivoli Dynamic Workload Broker server sends a job execution request to its agent on computer AIX-B where the agent submits the job to AIX.

To move an existing Tivoli Workload Scheduler for z/OS end-to-end job to be submitted by the Tivoli Dynamic Workload Broker you typically perform the following steps:

1. Collect Tivoli Workload Scheduler for z/OS end-to-end job information.
   a. Identify the command or script that is executed by the job. This information is found either in the SCRTLIB member (if it is a non-centralized script type job) or in a joblib member (if it is a centralized script type job).
   b. Identify the job stream and job name. (Keep in mind that there might be more than one.)
   c. Identify the current job duration time.
   d. Identify internal and external job dependencies.
2. Collect Tivoli Dynamic Workload Broker job information.
   a. Identify the job resource requirements (CPU architecture, memory, operating system, logical resources).
   b. Identify computer resources (managed by Tivoli Dynamic Workload Broker) that will be candidates for job resource allocation.
3. Create a Tivoli Dynamic Workload Broker job brokering definition.

   Select a suitable Tivoli Dynamic Workload Broker job brokering name. Keep in mind that the name is case sensitive.

   Chapter 11, “Managing Tivoli Dynamic Workload Broker jobs using Tivoli Workload Scheduler for z/OS end-to-end” on page 515, describes in detail how to create job brokering definitions.
4. Create a new Tivoli Workload Scheduler for z/OS end-to-end job stream.

   This can be a copy of the current job stream, but with a new job definition running on the Tivoli Dynamic Workload Broker workstation. Keep in mind that you must define the new job as a non-centralized script type job.

   Create a new SCRPTLIB member.
   a. Create a SCRIPTLIB member with a name that equals the TWS jobname specified in the job stream.
   b. Add a JOBREC keyword.
   c. Add a JOBCMD('<job-brokering-name>') parameter. Keep in mind that Tivoli Dynamic Workload Broker job brokering names are case sensitive. Also add Tivoli Dynamic Workload Broker job definition variable overrides as appropriate.
   d. Save the SCRPTLIB member.

Restriction: The Job Brokering Definition Console has import/export functionality that aids in migrating Tivoli Workload Scheduler jobs to the Dynamic Workload Broker. However, this Job Brokering Definition Console functionality cannot be used when migrating jobs from the Tivoli Workload Scheduler for z/OS end-to-end.

11.5 Planning and choreography advanced topics

In this section we focus on integration considerations for the following advanced planning and choreography topics:

- Logical resource usage and scope
  - Tivoli Dynamic Workload Broker logical resources
  - Tivoli Dynamic Workload Broker group resources
  - Tivoli Workload Scheduler for z/OS special resources
  - Sample resource usage scenario
- Serializing access to resources
- Job affinity definition
- Job recovery and restart
- Job tailoring using variables
  - Tivoli Workload Scheduler for z/OS variables in SCRPTLIB
  - Tivoli Dynamic Workload Broker variables in JSDL
  - Sample variables usage scenario

Note: In Tivoli Workload Scheduler for z/OS end-to-end the terms *job stream* and *application description* (AD) are synonyms. In this section we have chosen only to use the term *job stream*.

11.5.1 Logical resource usage and scope

Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end both allow you to choreograph job scheduling using the concept of logical resources. We first briefly describe the resource concepts used in the two products and then we describe a sample usage scenario.

Tivoli Dynamic Workload Broker resources

The Tivoli Dynamic Workload Broker has two resource concepts, *Logical Resources* and *Group Resources*. We only give a brief description here. Refer to Chapter 11, “Managing Tivoli Dynamic Workload Broker jobs using Tivoli Workload Scheduler for z/OS end-to-end” on page 515, for a more in-depth description of the concepts.

Logical Resources

Tivoli Dynamic Workload Broker logical resources are typically use to represent some computer characteristic that is not automatically discovered by the Tivoli Dynamic Workload Broker agents, such as software licenses or installed applications.

Logical resources are defined as being associated with one or more of the computers known by the Tivoli Dynamic Workload Broker. For example, you might create a logical resource named DB2 and add it to all the workstations where DB2 is installed.

Tivoli Dynamic Workload Broker jobs can be defined requiring one or more logical resources. For example, any job requiring logical resource DB2 would be automatically assigned to one of the computers that have this logical resource associated to it.

Group Resources

Tivoli Dynamic Workload Broker group resources are typically used to represent groups of computers with similar hardware, software, or other logical characteristics. For example, you might create a group resource named research and add to it all the computers in the research department.

Tivoli Dynamic Workload Broker jobs can be defined requiring a group resource. For example, any job requiring group resource *research* would be automatically assigned to one of the computers that have this group resource associated to it.

Tivoli Workload Scheduler for z/OS special resources

Tivoli Workload Scheduler for z/OS has the *special resources* concept. These special resources are typically use to represent some job requirement such as software licenses or application.

A feature of Tivoli Workload Scheduler for z/OS special resources is the availability status (can be yes or no). Tivoli Workload Scheduler for z/OS provides advanced mechanisms for event-based handling of the special resource availability status. A common usage of special resource availability status is to control when jobs are allowed to run during the day. Any job requiring a special resource is only eligible for submission if the special resource availability status is yes.

Resource scope

When working with planning and choreography in a Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end integrated environment it is important to keep in mind that the resource concepts have different job scheduling scopes. By scope we mean what jobs that can use, and be controlled by, a resource requirement.

The scope of Tivoli Dynamic Workload Broker logical resources and group resources is the jobs being managed by the Tivoli Dynamic Workload Broker server. The server makes scheduling decisions based on job resource requirements and availability of logical resources or group resources.

The scope of Tivoli Workload Scheduler for z/OS end-to-end special resources is the jobs being managed by the Tivoli Workload Scheduler for z/OS end-to-end engine. The Tivoli Workload Scheduler engine makes scheduling decisions based on job resource requirements and availability of special resources. The special resource concept is broader in scope because it can be used for any workstation and for any type of job.

11.5.2 Sample resource usage scenario

In our sample usage scenario we look at a fictive company that has integrated Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end. The company has three main business areas (A, B, and C), and each area has batch workloads running on distributed servers. Job scheduling to the distributed servers is dynamically managed using the Tivoli Dynamic Workload Broker. Business area A also has a large amount of mainframe jobs. The scenario is shown in Figure 11-18.

Note: The purpose of the scenario is to illustrate how the resource concepts can be used. Some user interface samples are included, but detailed step-by-step user interface actions have been deliberately left out.

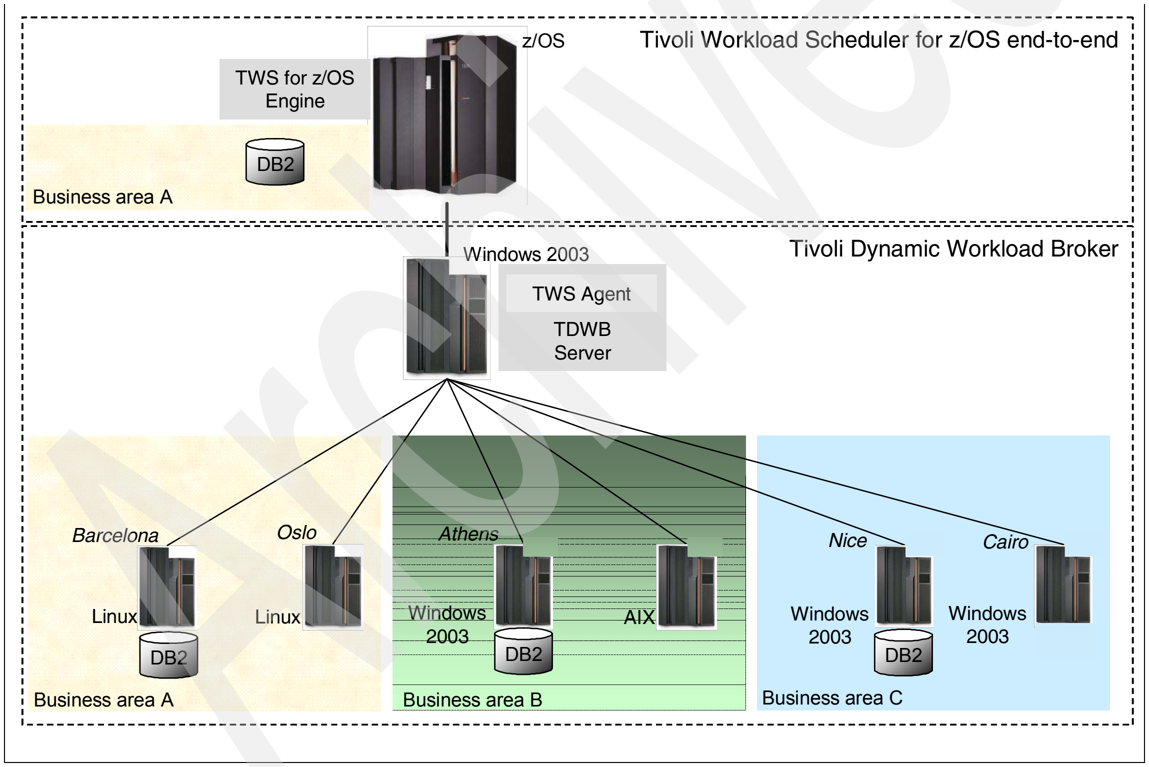


Figure 11-18 Business configuration input to resource choreography planning

Let us see how the company staff responsible for planning and choreography has used the resource concepts available to address the following requirements:

- Requirement 1

  Business area A distributed jobs must only be allowed to run on computers belonging to this business area (currently computers Barcelona and Oslo).

  Correspondingly business area B distributed jobs must only be allowed to run on computer Athens, and business area C distributed jobs must only be allowed to run on computers Nice and Cairo.

- Requirement 2

  Certain jobs require access to DB2 software. These jobs must only run on computers were DB2 is installed and ready to use. This requirement applies to all business areas. Also, several DB2 jobs must be allowed to run in parallel.

- Requirement 3

  Business area A uses online applications located on the mainframe (CICS® transaction server) and on Linux machine Barcelona (Web server). Both online applications access data stored in DB2.

  The DB2 data is analyzed by business analysts submitting *ad hoc* batch jobs, both on mainframe and on Linux servers. Some of these jobs are resource-intensive DB2 SQL queries. During open office hours online response time requirements are critical, so the resource-intensive DB2 query jobs must not be allowed to run. Outside open office hours the jobs are allowed to run and also there is currently no need to restrict how many jobs are run in parallel.

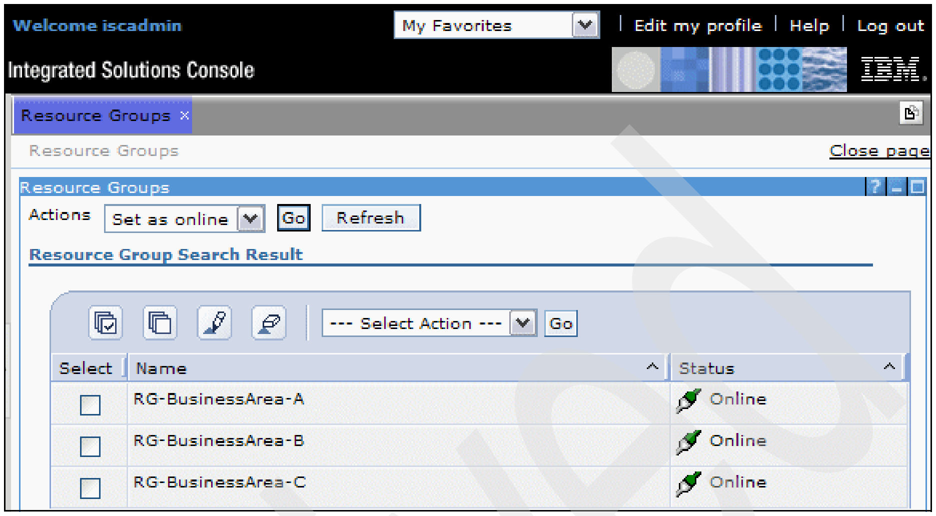
Solution for requirement 1

This solution is based on *Tivoli Dynamic Workload Broker group resources*. Three group resources are defined, as shown in Table 11-1.

Table 11-1 Tivoli Dynamic Workload Broker group resource setup

Group resource name	Associated computers
RG-BusinessArea-A	Barcelona, Oslo
RG-BusinessArea-B	Athens
RG-BusinessArea-C	Nice, Cairo

Figure 11-19 shows the group resource definitions in the Web Console.



Select	Name	Status
☐	RG-BusinessArea-A	Online
☐	RG-BusinessArea-B	Online
☐	RG-BusinessArea-C	Online

Figure 11-19 Tivoli Dynamic Workload Broker Web Console Group Resource list

All broker managed jobs in business area A are defined with a requirement for group resource RG-BusinessArea-A. Broker managed jobs in business areas B and C are defined correspondingly. The group resource requirement is added to the job brokering JSDL definition, as shown in Example 11-9. In the example the job requires group resource *RG-BusinessArea-C*, which ensures that only computers in business area C are candidates for running the job.

Example 11-9 Group resource requirement added to JSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
description="Business area C Windows job" name="BA-C-Windows-job">
  <jsdl:application name="executable">
    <jsdle:executable>
      <jsdle:script>echo "Windows job is running"</jsdle:script>
    </jsdle:executable>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:group name="RG-BusinessArea-C"/>
  </jsdl:resources>
</jsdl:jobDefinition>
```

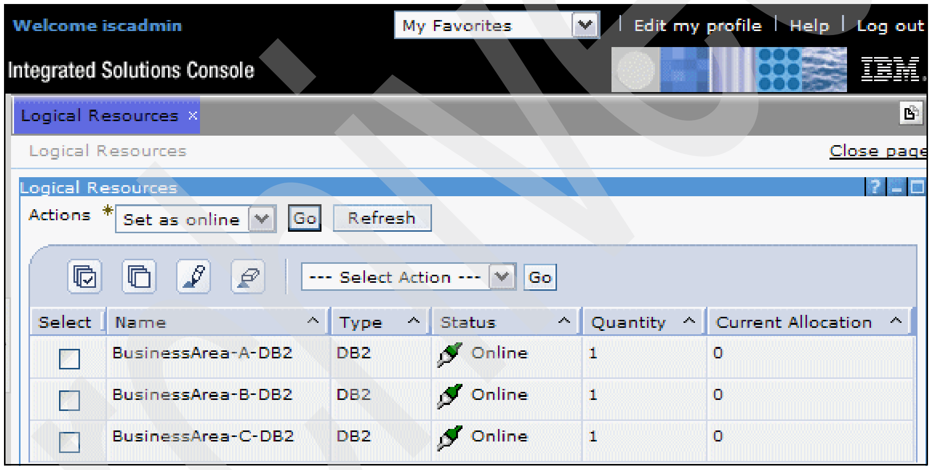
Solution for requirement 2

This solution is based on *Tivoli Dynamic Workload Broker logical resources*. Three logical resources are defined in Table 11-2.

Table 11-2 Tivoli Dynamic Workload Broker logical resource setup

Name	Type	Quantity	Associated computers
BusinessArea-A-DB2	DB2	1	Barcelona
BusinessArea-B-DB2	DB2	1	Athens
BusinessArea-C-DB2	DB2	1	Nice

Figure 11-20 shows the logical resource definitions in the Web Console.



Select	Name	Type	Status	Quantity	Current Allocation
☐	BusinessArea-A-DB2	DB2	Online	1	0
☐	BusinessArea-B-DB2	DB2	Online	1	0
☐	BusinessArea-C-DB2	DB2	Online	1	0

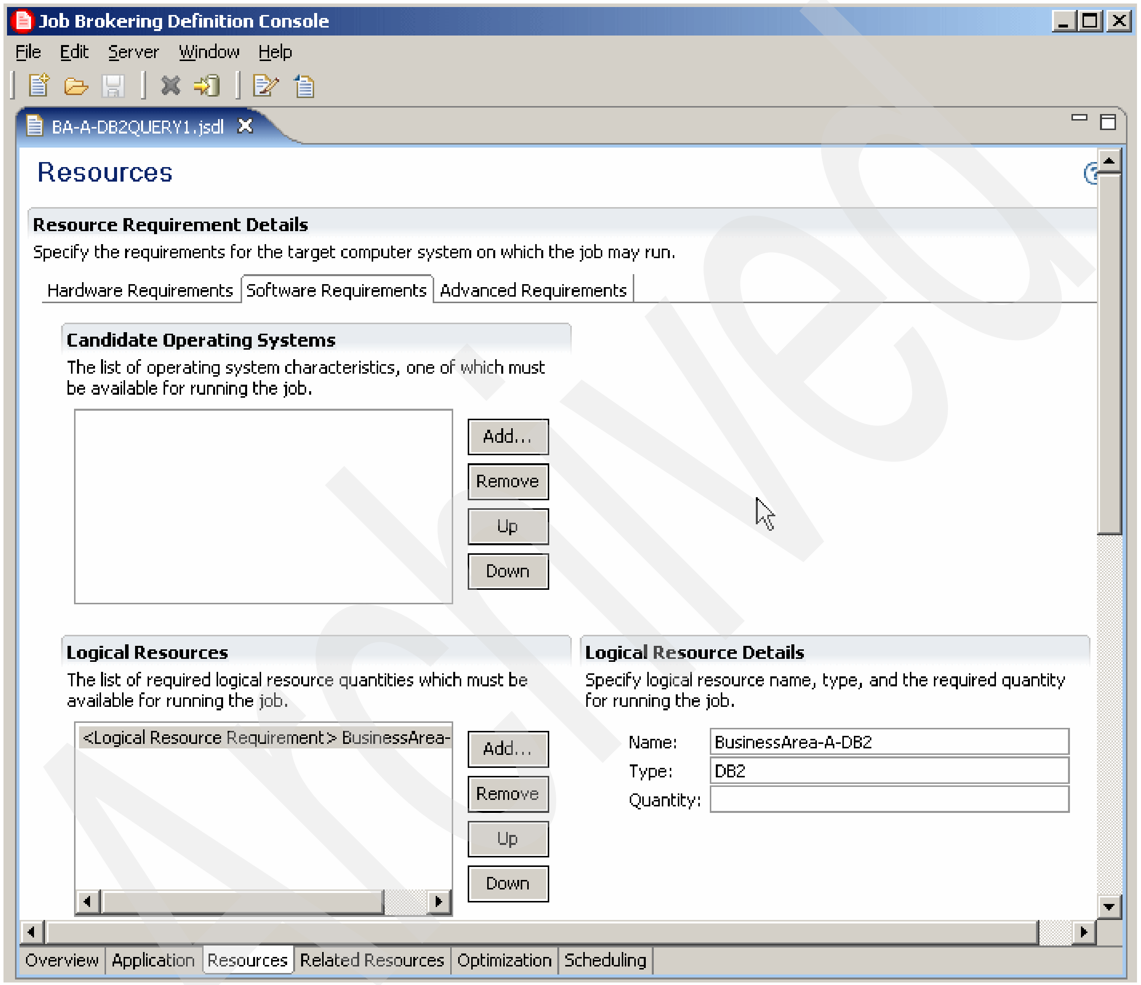
Figure 11-20 Logical resources list in Tivoli Dynamic Workload Broker Web Console

All broker managed DB2 jobs in business area A are defined with a requirement for logical resource BusinessArea-A-DB2. Broker managed jobs in business areas B and C that require DB2 are defined correspondingly. The logical resource requirement is added to the job brokering JSDL definitions, as shown in Example 11-10. In the example the job requires both logical resource *BusinessArea-A-DB2* and group resource *RG-BusinessArea-A*, which ensures that only computers in business area A with DB2 installed are candidates for running the job.

Example 11-10 Defining JDSL job definition with both logical and group resource

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
description="Business area A  DB2 query job" name="BA-A-DB2QUERY1">
  <jsdl:application name="executable">
    <jsdle:executable>
      <jsdle:script>echo "DB2 query job is running"</jsdle:script>
    </jsdle:executable>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:logicalResource name="BusinessArea-A-DB2" subType="DB2"/>
    <jsdl:group name="RG-BusinessArea-A"/>
  </jsdl:resources>
</jsdl:jobDefinition>
```

Figure 11-21 shows how the logical resource requirement was added to a job using the Job Brokering Definition Console. In the Logical Resource Details section only logical resource Name and Type was specified. Logical resource Quantity was left blank, which will have the desired effect that several jobs can run in parallel using the resource as shared.



Specify the requirements for the target computer system on which the job may run.

The list of operating system characteristics, one of which must be available for running the job.

The list of required logical resource quantities which must be available for running the job.

<Logical Resource Requirement> BusinessArea-

Specify logical resource name, type, and the required quantity for running the job.

Name: BusinessArea-A-DB2
Type: DB2
Quantity:

Figure 11-21 Adding logical resource requirement using Job Brokering Definition Console

Solution for requirement 3

This solution is based on *Tivoli Workload Scheduler for z/OS special resources*. This was chosen because the requirement applies to both mainframe and distributed jobs.

A special resource is defined, as shown in Example 11-11.

Example 11-11 Tivoli Workload Scheduler for z/OS special resource setup

```
Special resource   : BUSINESS-AREA-A-DB2-QUERIES
Used For           : Planning and control
On Error           : Free
On Complete        : Assume system default
Defaults
  Quantity         : 1
  Available        : No
```

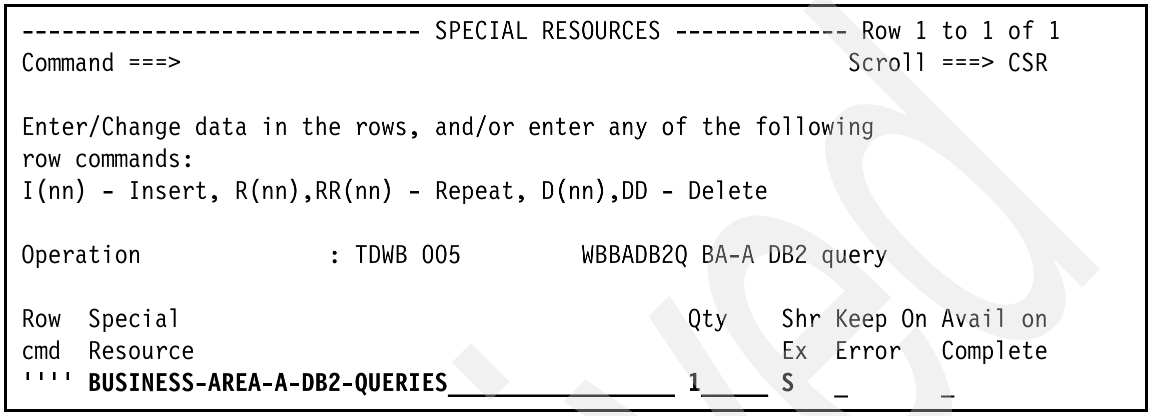
The special resource default availability is set to no. Availability status switching to Available=Yes after online office hours is controlled by a job stream that also closes down the CICS transaction server. The job stream contains two mainframe jobs, which runs in sequence:

- The first job causes the Tivoli Workload Scheduler for z/OS engine to send a signal to system automation software to close down the CICS transaction server. When close down is complete, system automation signals the Tivoli Workload Scheduler for z/OS engine to complete the job.
- The second job executes a mainframe job that by use of Tivoli Workload Scheduler for z/OS interfaces sends an SRSTAT event instructing the engine to change the special resource BUSINESS-AREA-A-DB2-QUERIES availability status to yes.

Availability switching to no at beginning of online office hours is controlled by a similar job stream that also starts up the CICS transaction server.

All resource-intensive DB2 query jobs, both mainframe and distributed, used by analysts in business area A are defined with a requirement for special resource BUSINESS-AREA-A-DB2-QUERIES. The special resource requirement is added to the jobs in the appropriate Tivoli Workload Scheduler for z/OS job stream definitions.

Figure 11-22 shows how the special resource was added to a job using TWS for z/OS ISPF panels SPECIAL RESOURCES. The special resource was added with Qty=1 (quantity) and Shr/Ex=S (allocation type Shared), which will have the desired effect that several DB2 jobs can run in parallel using the resource as shared.



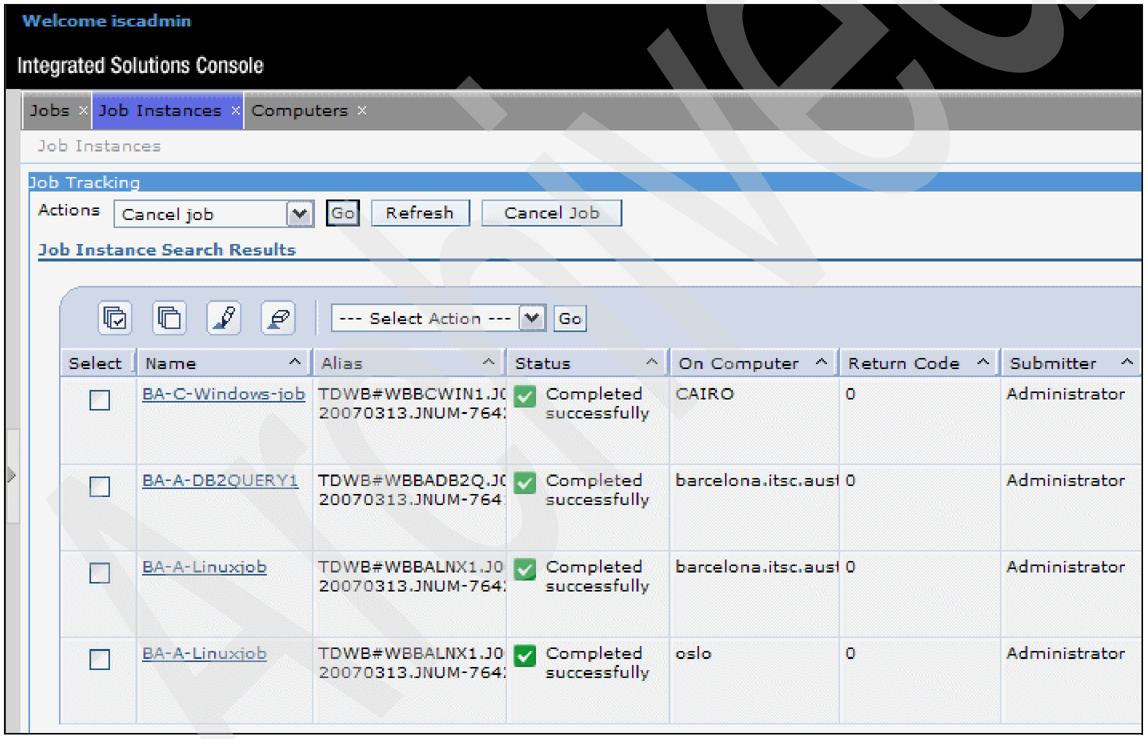
```
------------------------------ SPECIAL RESOURCES ------------- Row 1 to 1 of 1
Command ===>                                                  Scroll ===> CSR

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

Operation             : TDWB 005        WBBADB2Q BA-A DB2 query

Row  Special                                     Qty     Shr Keep On Avail on
cmd  Resource                                            Ex  Error   Complete
'''' BUSINESS-AREA-A-DB2-QUERIES_________________ 1_____ S   _       _
```

Figure 11-22   Special resource added to Tivoli Workload Scheduler end-to-end job

Solutions at work

Figure 11-23 shows the result of solutions one and two. Four sample jobs have been run. Tivoli Dynamic Workload Broker has allocated computers to the jobs using the logical resources and group resources requirements as follows:

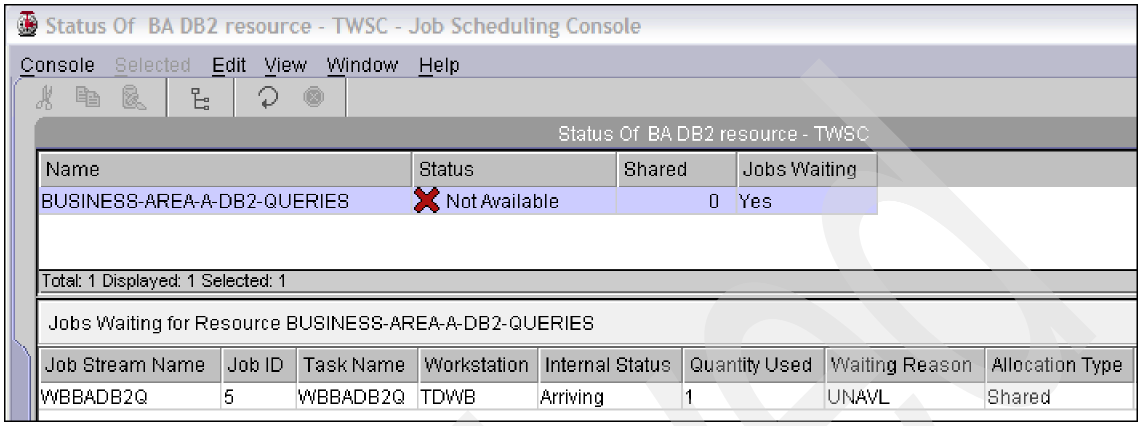
- Job *BA-C-Windows-job* requires group resource *RG-BusinessArea-C*. The job was allocated and run on computer Cairo.
- Job *BA-A-DB2QUERY1* requires logical resource *BusinessArea-A-DB2* and group resource *RG-BusinessArea-C*. The job was allocated and run on computer Barcelona.
- Job *BA-A-Linuxjob* requires group resource *RG-BusinessArea-A*. The job was run twice. In the first run the job was allocated and run on computer Barcelona. The second run was on computer Oslo.



Select	Name	Alias	Status	On Computer	Return Code	Submitter
☐	BA-C-Windows-job	TDWB#WBBCWIN1.J0 20070313.JNUM-764	✓ Completed successfully	CAIRO	0	Administrator
☐	BA-A-DB2QUERY1	TDWB#WBBADB2Q.J0 20070313.JNUM-764	✓ Completed successfully	barcelona.itsc.aust	0	Administrator
☐	BA-A-Linuxjob	TDWB#WBBALNX1.J0 20070313.JNUM-764	✓ Completed successfully	barcelona.itsc.aust	0	Administrator
☐	BA-A-Linuxjob	TDWB#WBBALNX1.J0 20070313.JNUM-764	✓ Completed successfully	oslo	0	Administrator

Figure 11-23 Job instance list in Tivoli Dynamic Workload Broker Web Console

Figure 11-24 shows the result of solution three. A screen sample from a Job Scheduling Console special resource plan list shows the status for special resource BUSINESS-AREA-A-DB2-QUERIES.


Status Of BA DB2 resource - TWSC - Job Scheduling Console
Console Selected Edit View Window Help
Status Of BA DB2 resource - TWSC

Name	Status	Shared	Jobs Waiting
BUSINESS-AREA-A-DB2-QUERIES	Not Available	0	Yes

Total: 1 Displayed: 1 Selected: 1

Jobs Waiting for Resource BUSINESS-AREA-A-DB2-QUERIES

Job Stream Name	Job ID	Task Name	Workstation	Internal Status	Quantity Used	Waiting Reason	Allocation Type
WBBADB2Q	5	WBBADB2Q	TDWB	Arriving	1	UNAVL	Shared

Figure 11-24 Job waiting for TWS for z/OS special resource

The top part shows that special resource BUSINESS-AREA-A-DB2-QUERIES is in status *Not available* and that the resource has jobs waiting for it (*Yes* in *the* Jobs Waiting column).

The bottom part shows a job (TWS job WBBADB2Q in job stream with same name) waiting for the special resource. The waiting reason is *UNAVL*, because the special resource status is *Not available*. The job is defined to workstation TDWB, which in our scenario means that this is a Tivoli Dynamic Workload Broker job.

This situation is an example of what would occur if, for example, a developer in business area A adds a DB2 query job stream during online office hours.

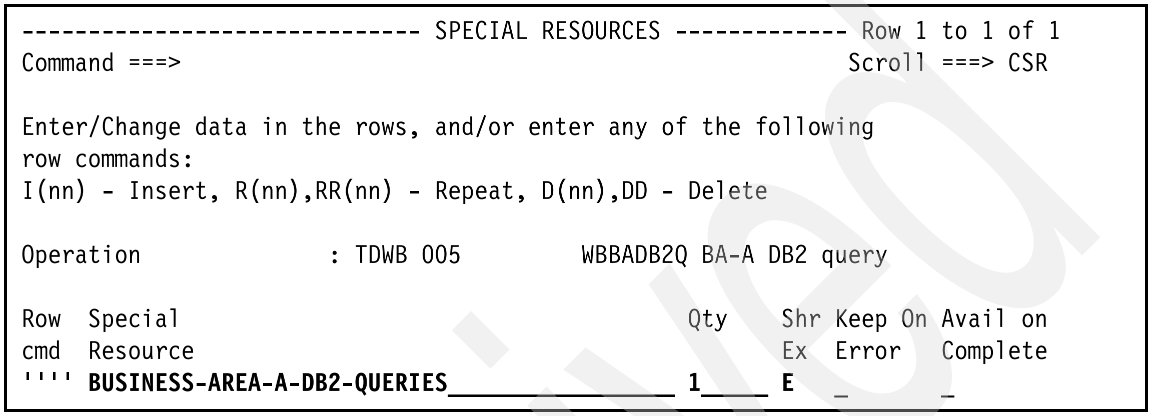
11.5.3 Serializing access to resources

Both Tivoli Dynamic Workload Broker *logical resources* and Tivoli Workload Scheduler for z/OS *special resources* can be used to address the need for serializing access to resources.

Let us continue looking at the fictive company configuration shown in Figure 11-18 on page 568 in 11.5.2, “Sample resource usage scenario” on page 568.

Business areas A and C both need a solution for a new business requirement: DB2 jobs must no longer be allowed to run in parallel.

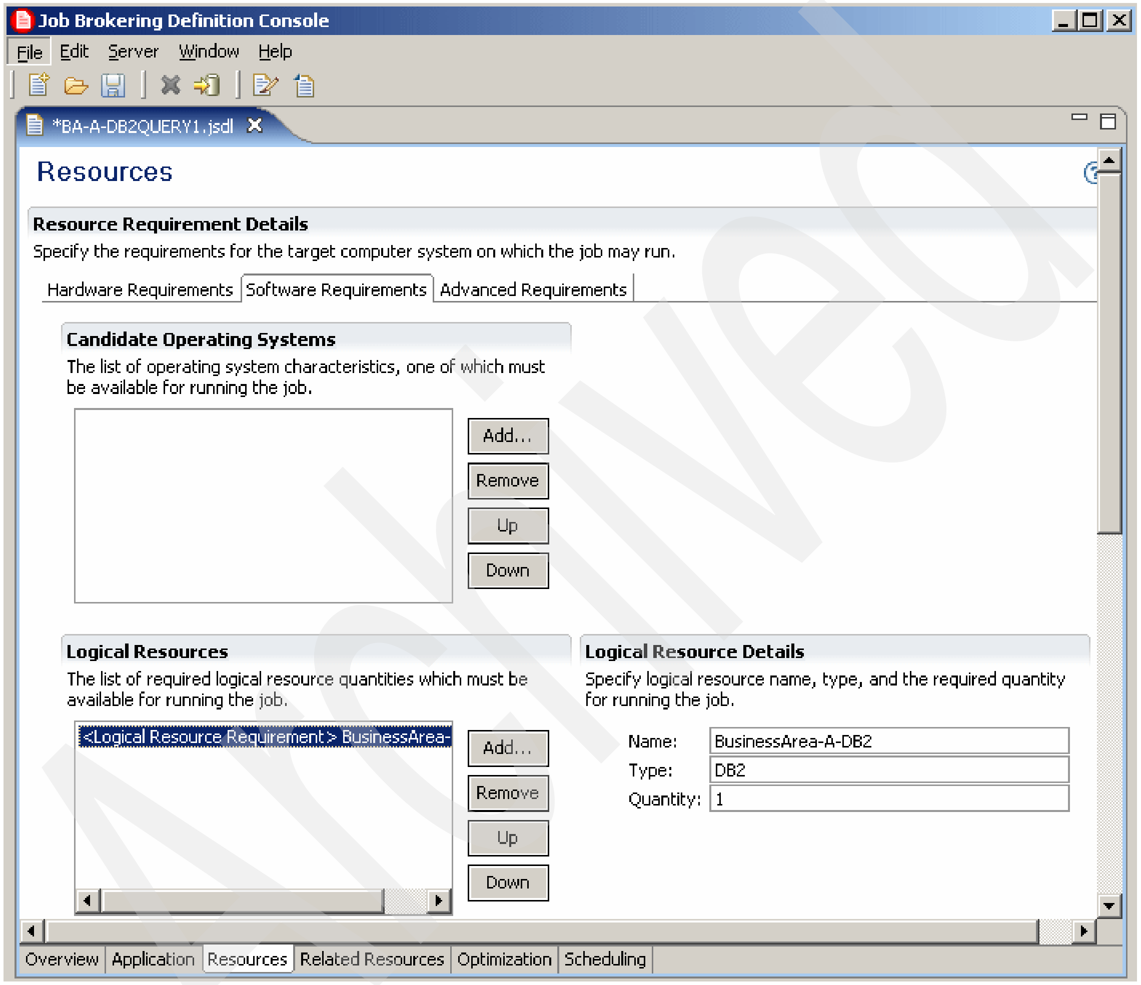
The solution for business area A is based on Tivoli Workload Scheduler for z/OS *special resources*. The Tivoli Workload Scheduler for z/OS job stream definitions are modified for the DB2 jobs in business area A, as shown in Figure 11-25. The special resource allocation is changed to Shr/Ex=E (allocation type Exclusive), which will have the desired effect that only one DB2 job at a time can run using the resource as exclusive.



```
------------------------------ SPECIAL RESOURCES ------------- Row 1 to 1 of 1
Command ===>                                                  Scroll ===> CSR

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

Operation             : TDWB 005        WBBADB2Q BA-A DB2 query

Row  Special                                 Qty    Shr Keep On Avail on
cmd  Resource                                       Ex  Error   Complete
'''' BUSINESS-AREA-A-DB2-QUERIES_________________ 1_____ E   _       _
```

Figure 11-25 Special resource allocation type exclusive

The solution for business area C is based on Tivoli Dynamic Workload Broker *logical resources*. The Tivoli Dynamic Workload Broker job brokering definitions are modified for the DB2 jobs in business area C, as shown in Figure 11-26. The logical resource quantity is specified to 1, which will have the desired effect that only one job at a time can run using the resource as exclusive.


Job Brokering Definition Console
File Edit Server Window Help
*BA-A-DB2QUERY1.jsdl
Resources
Resource Requirement Details
Specify the requirements for the target computer system on which the job may run.
Hardware Requirements
Software Requirements
Advanced Requirements
Candidate Operating Systems
The list of operating system characteristics, one of which must be available for running the job.
Add...
Remove
Up
Down
Logical Resources
The list of required logical resource quantities which must be available for running the job.
<Logical Resource Requirement> BusinessArea-
Logical Resource Details
Specify logical resource name, type, and the required quantity for running the job.
Overview Application Resources Related Resources Optimization Scheduling

Name: BusinessArea-A-DB2
Type: DB2
Quantity: 1

Figure 11-26   Adding logical resource requirement with quantity using JBDC

11.5.4 Job affinity definition

In Tivoli Dynamic Workload Broker, you define affinity relationships between two or more jobs when you want them to run on the same computer resource. There are two types of Tivoli Dynamic Workload Broker affinities:

- Tivoli Dynamic Workload Broker affinity
- Tivoli Workload Scheduler affinity

We discuss the Tivoli Workload Scheduler affinity relationship and how it is used in an environment where Tivoli Dynamic Workload Broker is integrated with Tivoli Workload Scheduler for z/OS end-to-end.

The Tivoli Dynamic Workload Broker affinity type is not covered here. It is described in detail in Chapter 5, "Advanced guide to Tivoli Dynamic Workload Broker" on page 175.

For the purpose of this discussion we assume that we have a job stream consisting of two jobs with TWS job names WBAFJ1 and WBAFJ2. The requirements are:

- The two jobs must run using Tivoli Dynamic Workload Broker to ensure that they are dynamically assigned to a computer with appropriate CPU resources.
- The two jobs must run in sequence, WBAFJ1 first and then WBAFJ2.
- The jobs must also run on the same computer, because analysis results generated by the first job are stored on the computer hard drive and must be available to the second job.

The solution for the first requirement is to define both jobs to run on a workstation that represents Tivoli Dynamic Workload Broker. In our case the workstation is TDWB, as shown in Example 11-12.

Example 11-12 TWS for z/OS job stream with two jobs

```
Application           : WBAFFINITY1      TDWB sample application

Row  Oper      Duration  Job name  Operation text
cmd  ws   no.  HH.MM.SS
'''' TDWB 005  00.00.01  WBAFJ1__  FTP get data and analyze
'''' TDWB 010  00.00.01  WBAFJ2__  Report generation_______
```

The solution for the second requirement is to define the first job as an internal job predecessor to the second job, as shown in Example 11-13.

Example 11-13 TWS for z/OS job stream with internal predecessor

```
Application           : WBAFFINITY1      TDWB sample application

Row  Oper      Duration  Job name  Internal predecessors
cmd  ws   no.  HH.MM.SS
'''' TDWB 005  00.00.01  WBAFJ1__  ___ ___ ___ ___ ___ ___ ___ ___
'''' TDWB 010  00.00.01  WBAFJ2__  005 ___ ___ ___ ___ ___ ___ ___
```

The solution for the third requirement is to add a Tivoli Dynamic Workload Broker *TWS affinity relationship* between the two jobs. This is done by defining the Tivoli Workload Scheduler for z/OS end-to-end SCRPTLIB members, as shown in Example 11-14.

Example 11-14 Defining TDWB job affinity between TWS for z/OS end-to-end jobs

```
EDIT       TWS.V8R30.SCRPTLIB(WBAFJ1) - 01.02              Columns 00001 00072
Command ===>                                                  Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 JOBREC
000002 JOBCMD('FTP-get-and-analyze')
****** **************************** Bottom of Data ****************************


EDIT       TWS.V8R30.SCRPTLIB(WBAFJ2) - 01.07              Columns 00001 00072
Command ===>                                                  Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 JOBREC
000002 JOBCMD('Report-generate -twsAffinity jobname=J005_WBAFJ1')
****** **************************** Bottom of Data ****************************
```

The JOBCMD() parameter for the second job contains two parts, the broker job name and the TWS affinity definition. The general syntax for the TWS affinity definition is:

```
-twsAffinity jobname=J<jobnumber>_<TWSjobName>
```

Where:

<jobnumber>	This is the operation sequence number (001–255) of the affinity relationship target job.
<TWSjobName>	This is the TWS for z/OS job name of the affinity relationship target job.

Important: The underscore character between the jobnumber and the TWSjobName is required. Also, if the JOBCMD() parameter includes more than one word, it must be enclosed within single or double quotation marks.

Restriction: Tivoli Workload Scheduler affinity relationships can only be used between jobs that are part of the same job stream.

11.5.5 Job recovery and restart

Tivoli Workload Scheduler for z/OS end-to-end includes choreography capabilities that allow you to prepare for job recovery and restart handling of Tivoli Dynamic Workload Broker jobs that end with a non-zero return code. Recovery and restart actions can include a recovery prompt, submission of a recovery job, or both.

- Recovery and restart options and actions are defined in the Tivoli Workload Scheduler for z/OS end-to-end SCRPTLIB member.
- Manual recovery and restart actions are performed using Tivoli Workload Scheduler for z/OS end-to-end user interfaces.
- Automatic recovery and restart actions are performed by Tivoli Workload Scheduler for z/OS end-to-end.

Let us look at a simple example to illustrate some of the recovery and restart choreography options available. For the purpose of our discussion we assume that job stream name WBERRORTEST3 with job WBFAILUR has been defined. Example 11-15 shows SCRPTLIB member WBFAILUR that instructs Tivoli Workload Scheduler for z/OS Scheduling end-to-end to run a Tivoli Workload Broker job.

Example 11-15 Defining recovery and restart options in SCRPTLIB member

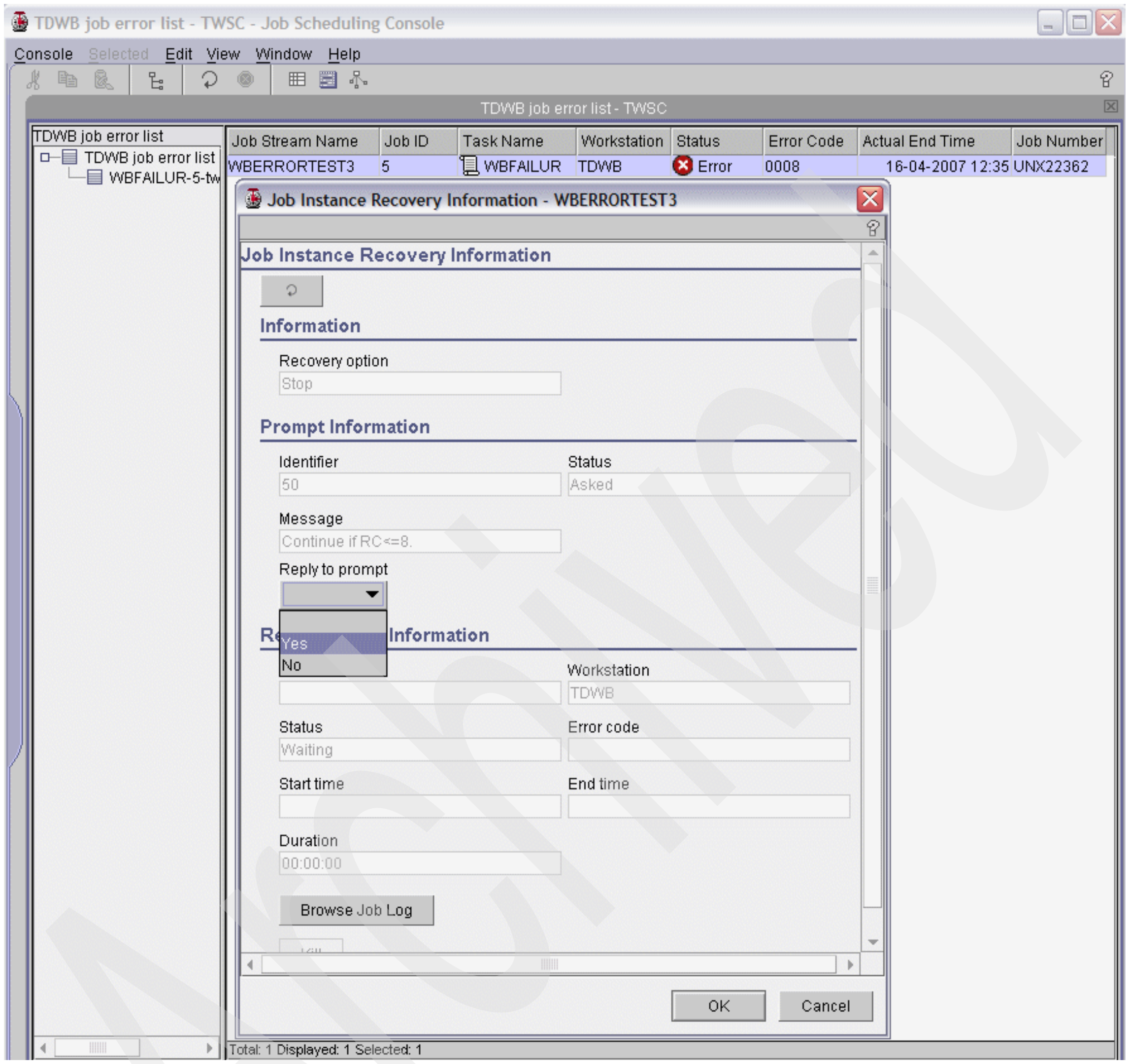
```
JOBREC
JOBCMD('twsz-UNIX-script')
RECOVERY OPTION(STOP)
MESSAGE('Continue if RC<=8.')
JOBCMD('twsz-UNIX-recovery-script')
```

The SCRPTLIB statements used are JOBREC and RECOVERY. When job stream WBERRORTEST3 is run from the current plan the following sequence of events occur:

1. Tivoli Workload Scheduler for z/OS end-to-end submits job WBFAILUR to Tivoli Dynamic Workload Broker.
2. Tivoli Dynamic Workload Broker runs job twsz-UNIX-script and the execution result is returned to Tivoli Workload Scheduler for z/OS end-to-end.
3. If job WBFAILUR ends in complete status, then no recovery or restart actions are performed. Scheduling of successors (if there are any) continues.
4. If job WBFAILUR ends in error status:
   a. Tivoli Workload Scheduler for z/OS end-to-end waits for manual intervention.
   b. When manual recovery is invoked a recovery prompt with message text`'Continue if RC<=8.'` is displayed.
   c. If the the reply to recovery prompt is no, no further recovery is performed. The job WBFAILUR remains in error status.
   d. If the reply to the recovery prompt is yes, Tivoli Workload Scheduler for z/OS end-to-end submits a recovery job to the Tivoli Dynamic Workload Broker. In out example the recovery job definition instructs Tivoli Dynamic Workload Broker to run job **`twsz-UNIX-recovery-script`**.
   e. Tivoli Workload Broker runs job twsz-UNIX-recovery-script and the execution result is returned to Tivoli Workload Scheduler for z/OS end-to-end.
   f. If the recovery job is successful, job WBFAILUR is set to completed status and scheduling of successors (if there are any) continues.
   g. If the recovery job ends in error status, no further recovery is performed. Job WBFAILUR remains in error status.

Figure 11-27 on page 584 shows how a trial run of job stream WBERRORTEST3 is in the process of being handled by an operations analyst using the Job Scheduling Console. The figure shows:

- The Job Scheduling Console window.
- Job WABFAILUR in status error with error code 008 (blue shaded line at top of figure).
- The Recovery Information window (in center of figure).
- The Operations analyst is about to select yes as reply to PROMPT Message `Continue if RC<=8`.


TDWB job error list - TWSC - Job Scheduling Console
Console Selected Edit View Window Help
TDWB job error list - TWSC
TDWB job error list
TDWB job error list
WBFAILUR-5-tw

Job Stream Name	Job ID	Task Name	Workstation	Status	Error Code	Actual End Time	Job Number
WBERRORTEST3	5	WBFAILUR	TDWB	Error	0008	16-04-2007 12:35	UNX22362

Job Instance Recovery Information - WBERRORTEST3

Job Instance Recovery Information

Information

Recovery option: Stop

Prompt Information

Identifier: 50
Status: Asked
Message: Continue if RC<=8.
Reply to prompt: Yes / No

Information

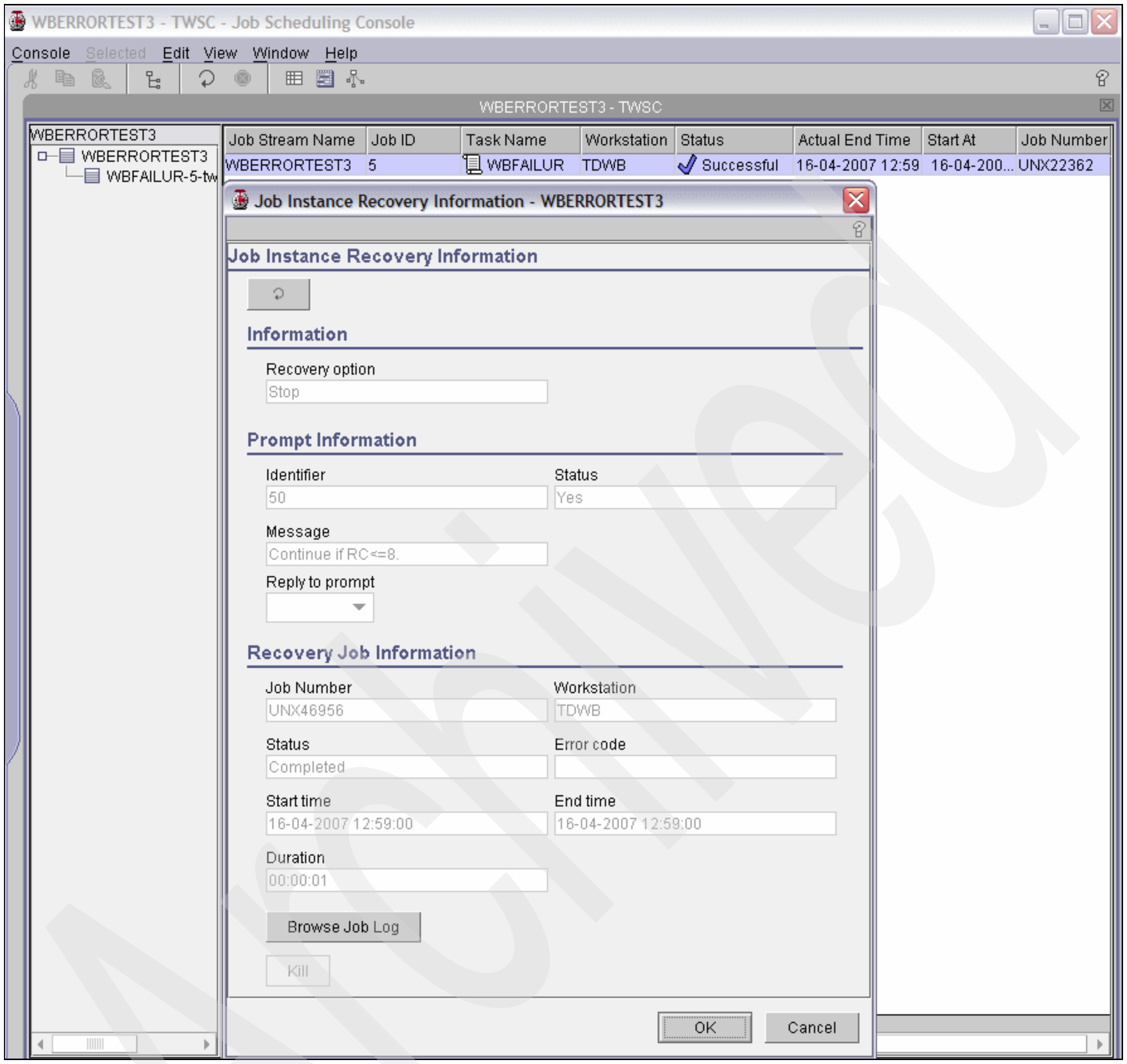
Workstation: TDWB
Status: Waiting
Error code:
Start time:
End time:
Duration: 00:00:00

Total: 1 Displayed: 1 Selected: 1

Figure 11-27 Job Instance Recovery information - recovery prompt

Figure 11-28 on page 586 shows the situation after the recovery job has run successfully. The operations analyst is using the Job Scheduling Console to verify the results of recovery actions. The figure shows:

- The Job Scheduling Console window.
- Job WABFAILUR in status successful (blue shaded line at top of figure).
- The Recovery Information window (in center of figure).
- Recovery Job Information shows that the recovery job (with job number UNX16956) ended in status completed and that it was run at workstation TDWB.


WBERRORTEST3 - TWSC - Job Scheduling Console

Job Stream Name	Job ID	Task Name	Workstation	Status	Actual End Time	Start At	Job Number
WBERRORTEST3	5	WBFAILUR	TDWB	✓ Successful	16-04-2007 12:59	16-04-200...	UNX22362

Job Instance Recovery Information - WBERRORTEST3

Job Instance Recovery Information

Information

Recovery option: Stop

Prompt Information

Identifier: 50

Status: Yes

Message: Continue if RC<=8.

Reply to prompt:

Recovery Job Information

Job Number: UNX46956

Workstation: TDWB

Status: Completed

Error code:

Start time: 16-04-2007 12:59:00

End time: 16-04-2007 12:59:00

Duration: 00:00:01

Figure 11-28 Job Instance Recovery information - recovery completed

11.5.6 Job tailoring using variables

Job tailoring enable jobs to be automatically edited using information that is known only at job setup or submit. This can reduce your dependency on time-consuming and error-prone manual editing of jobs. Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end both allow you to choreograph job scheduling using automatic variable substitution in job definitions.

In this section we use a sample scenario to illustrate how the job tailoring capability of both products can be used and how the two products can complement each other.

Note: The purpose of the scenario is to illustrate how variables can be used. Some user interface samples are included, but detailed step-by-step user interface actions have been deliberately left out.

First we briefly describe the variable constructs that are used in our sample scenario and then we continue with a step-by-step description of the scenario itself.

Tivoli Workload Scheduler for z/OS variables in SCRPTLIB

Tivoli Workload Scheduler for z/OS end-to-end has several job tailoring capabilities. For the purpose of our discussion we focus on the usage of variables in SCRPTLIB members.

The content of a sample SCRPTLIB member is shown in Example 11-16, where variable &OADID is used in the JOBCMD() parameter.

Example 11-16 Using variable &OADID in SCRPTLIB member

```
VARSUB
JOBREC
JOBCMD('twsz-tdwbcli-cmd -var tdwbcmdparms=-alias "TDWB#&OADID*"')
```

The variable &OADID is a so-called supplied variable, and it will be automatically substituted with the name of the job stream occurrence containing the job. Tivoli Workload Scheduler for z/OS supplies variables that you can use in your business, and &OADID is one of them. Besides the supplied variables, you can also define your own installation-specific variables, which are termed user-defined variables.

Substitution of variables in SCRPTLIB members occurs when:

- The daily planning process creates the symphony file.
- The job stream is added to the plan manually.

Refer to *IBM Tivoli Workload Scheduler for z/OS Scheduling End-to-end,* SC32-1732, for full description of syntax, substitution rules, and variable usage in SCRPTLIB members.

Refer to *IBM Tivoli Workload Scheduler for z/OS Managing the Workload,* SC32-1263, for a full description of supplied and user-defined variables.

Tivoli Dynamic Workload Broker variables in JSDL

In a JSDL definition, you can define variables that associate a symbolic name with a value. The variables defined can be referenced in values elsewhere in the JSDL definition using the format ${variable_name}.

Example 11-17 shows an example of variables definition and reference in a JDSL. The relevant statements are marked with bold text.

Example 11-17 Using variables in JSDL definition

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" description="Sample
TDWB Windows job executing the TDWB CLI" name="twsz-tdwbcli-cmd">
  <jsdl:annotation>This job can run on Windows  where TDWB server is
installed</jsdl:annotation>
  <jsdl:variables>
    <jsdl:stringVariable name="tdwbcmd">"C:\Program
Files\IBM\ITDWB\Server\bin\jobquery.bat"</jsdl:stringVariable>
    <jsdl:stringVariable name="tdwbcmdparms">-name "tws*"</jsdl:stringVariable>
  </jsdl:variables>
  <jsdl:application name="executable">
    <jsdle:executable path="C:\utilities\tdwbcli.cmd" workingDirectory="c:\">
      <jsdle:arguments>
        <jsdle:value>${tdwbcmd} ${tdwbcmdparms}</jsdle:value>
      </jsdle:arguments>
    </jsdle:executable>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:candidateHosts>
      <jsdl:hostName>athens</jsdl:hostName>
    </jsdl:candidateHosts>
  </jsdl:resources>
</jsdl:jobDefinition>
```

In the example two variables are defined with names and default values shown in Table 11-3.

Table 11-3 JSDL variable name and default values

Variable name	Default value
tdwbcmd	“C:\Program Files\IBM\ITDWB\Server\bin\jobquery.bat”

Variable name	Default value
tdwbcmdparms	-name "tws*"

In Example 11-17 on page 588 the two variables are referenced in the part of the JDSL that defines the program executable and the parameters to be passed to it. The *value* element is defined as ${tdwbcmd} ${tdwbcmdparms}. The resulting command that is built (using default variable values) by Tivoli Dynamic Workload Broker and used to start job execution is shown in Example 11-18.

Example 11-18 TDWB job start command resulting from variables substitution

```
C:\utilities\tdwbcli.cmd
“C:\Program Files\IBM\ITDWB\Server\bin\jobquery.at” -name "tws*"
```

Substitution of variables in JSDL is done by the Tivoli Dynamic Workload Broker server and occurs when the job is submitted. JSDL variable values can be overridden when the jobsubmit command is issued to the Tivoli Dynamic Workload Broker server. In the scenario that follows we use this overriding mechanism when Tivoli Workload Scheduler for z/OS end-to-end submits job requests to the Tivoli Dynamic Workload Broker server.

Refer to the *Tivoli Dynamic Workload Broker User’s Guide* for a more detailed description of Tivoli Dynamic Workload Broker variables. Variables are also described in Chapter 5, “Advanced guide to Tivoli Dynamic Workload Broker” on page 175.

In our sample scenario that follows we show you how to define and use variables in JSDL using the Job Brokering Definition Console.

11.5.7 Sample variables usage scenario

In our sample scenario we have a fictive company that has integrated Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end. Job scheduling to distributed servers is dynamically managed using the Tivoli Dynamic Workload Broker. The Tivoli Dynamic Workload Broker server is running on a Windows 2003 computer named Athens. The Athens computer also has a Tivoli Dynamic Workload Broker agent installed.

Business need

Let us see how the company staff responsible for planning and choreography uses variables as part of a solution that addresses the following needs:

- Operation analysts would like a more effective way to retrieve and access information about jobs that have been run by the Tivoli Dynamic Workload Broker. Currently, they log on to the computer where the broker server is running and issue Tivoli Dynamic Workload Broker command-line interface jobquery commands.
- Operations analysts would also like to issue the queries and get the results while using the Tivoli Workload Scheduler for z/OS user interfaces.
- If possible, queries should be run automatically when all jobs in certain job streams have been run.

Overview of solution components

The solution components; are:

- A Windows command file. The purpose of this command file is to issue a Tivoli Dynamic Workload Broker CLI command. The CLI command to be issued (and its parameters) is delivered to the command file as parameters.
- A Tivoli Dynamic Workload Broker job that executes the Windows command file.
- Tivoli Dynamic Workload Broker JSDL variables are used when specifying the Windows command file parameters in the JDSL.
- A Tivoli Workload Scheduler for z/OS end-to-end job that runs the JSDL job. The end-to-end job is defined so that it specifies overriding values for the JSDL variables.
- Tivoli Workload Scheduler for z/OS variables are used in the end-to-end job definition. The purpose is to achieve automatic insert of job stream occurrence information at the time that the end-to-end job is added to the current plan.

Solution implementation

We now describe the implementation of the four components one at a time.

Windows command file TDWBCLI.cmd

A Windows command file was created as follows:

File name	TDWBCLI.cmd
File location	C:\Utilities
File content	%*

A sample test (specifying `dir TDWBCLI*` as input) of the TDWBCLI.cmd from a command prompt is shown in Example 11-19.

Example 11-19 TDWBCLI.cmd executed from command prompt

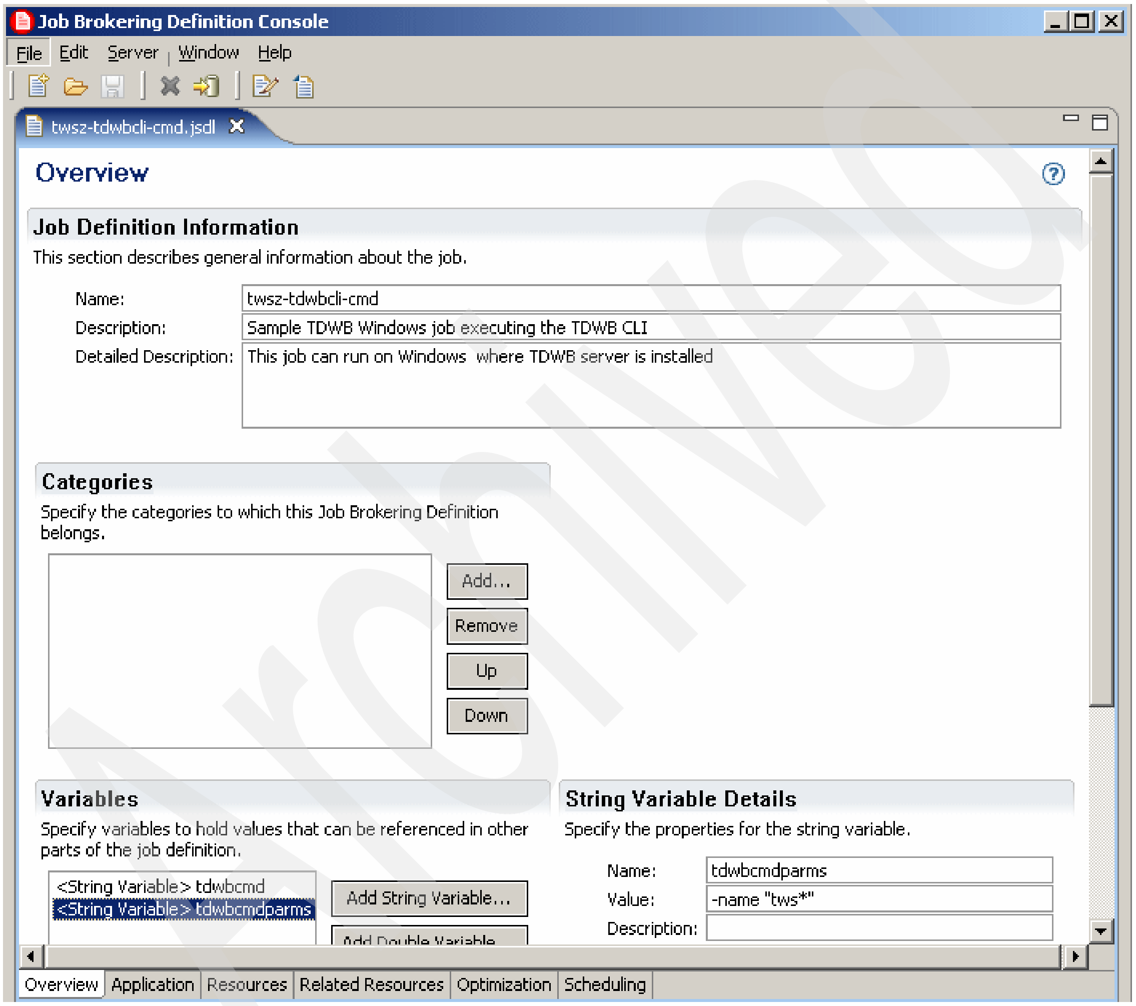
```
C:\Utilities>TDWBCLI.cmd dir TDWBCLI*
C:\Utilities>call dir TDWBCLI*
 Volume in drive C has no label.
 Volume Serial Number is D820-9A05

 Directory of C:\Utilities

03/30/2007  06:14 AM                 7 TDWBCLI.cmd
               1 File(s)             7 bytes
               0 Dir(s)  54,697,168,896 bytes free
```

Tivoli Dynamic Workload Broker job with variables

A JSDL job was created using the Job Brokering Definition Console.

Figure 11-29 shows how Job definition information was filled in with:

- Name, description, and detailed description.
- Two string variables, tdwbcmd and tdwbcmdparms. Variables are added by clicking **Add String Variable**.
- Variable tdwbcmdparms defined with default value `-name "tws*"`.


Job Brokering Definition Console
File Edit Server Window Help
twsz-tdwbcli-cmd.jsdl
Overview
Add...
Remove
Up
Down
Add String Variable...
Add Double Variable
Overview Application Resources Related Resources Optimization Scheduling

Job Definition Information

This section describes general information about the job.

Name:	twsz-tdwbcli-cmd
Description:	Sample TDWB Windows job executing the TDWB CLI
Detailed Description:	This job can run on Windows where TDWB server is installed

Categories

Specify the categories to which this Job Brokering Definition belongs.

Variables

Specify variables to hold values that can be referenced in other parts of the job definition.

<String Variable> tdwbcmd
<String Variable> tdwbcmdparms

String Variable Details

Specify the properties for the string variable.

Name:	tdwbcmdparms
Value:	-name "tws*"
Description:	

Figure 11-29 Name, description, and variables are added to Tivoli Dynamic Workload Broker job

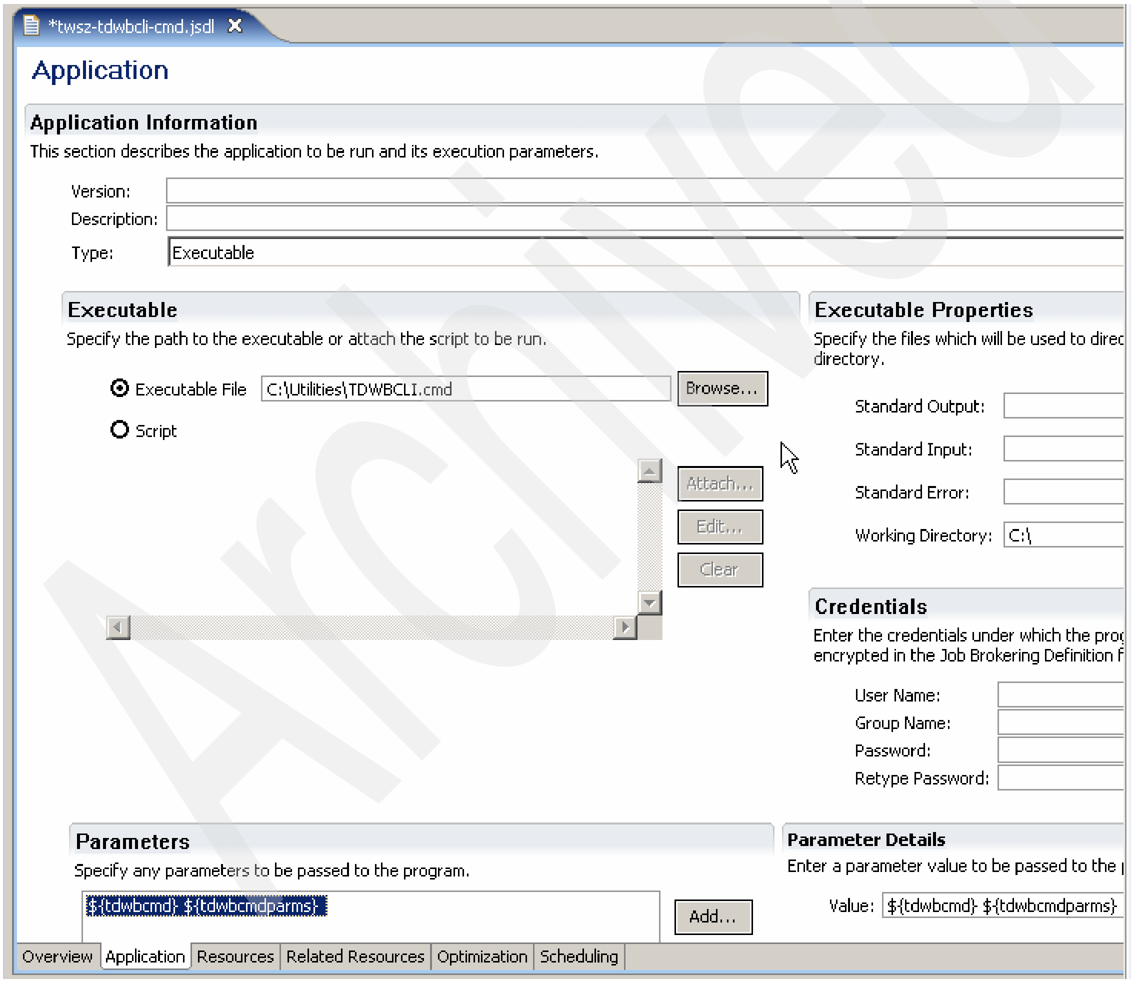
The variable tdwbcmd was defined with the default value:

```
"C:\Program Files\IBM\ITDWB\Server\bin\jobquery.bat"
```

The delimiting double quotes are part of the value. Why this value? Well, the first part, the path, is our Tivoli Dynamic Workload Broker installation directory containing the command-line interface executables, and the last part is the name of the **`jobquery`** command executable.

Next, application information was filled in, as shown in Figure 11-30, using the following information:

Application Type	Executable
Executable file	C:\Utilities\TDWBCLI.cmd
Working Directory	C:\
Parameter details	${tdwbcmd} ${tdwbcmdparms}



Application

Application Information

This section describes the application to be run and its execution parameters.

Version:

Description:

Type: Executable

Executable

Specify the path to the executable or attach the script to be run.

● Executable File C:\Utilities\TDWBCLI.cmd

○ Script

Executable Properties

Specify the files which will be used to direc directory.

Standard Output:

Standard Input:

Standard Error:

Working Directory: C:\

Credentials

Enter the credentials under which the prog encrypted in the Job Brokering Definition f

User Name:

Group Name:

Password:

Retype Password:

Parameters

Specify any parameters to be passed to the program.

${tdwbcmd} ${tdwbcmdparms}

Parameter Details

Enter a parameter value to be passed to the

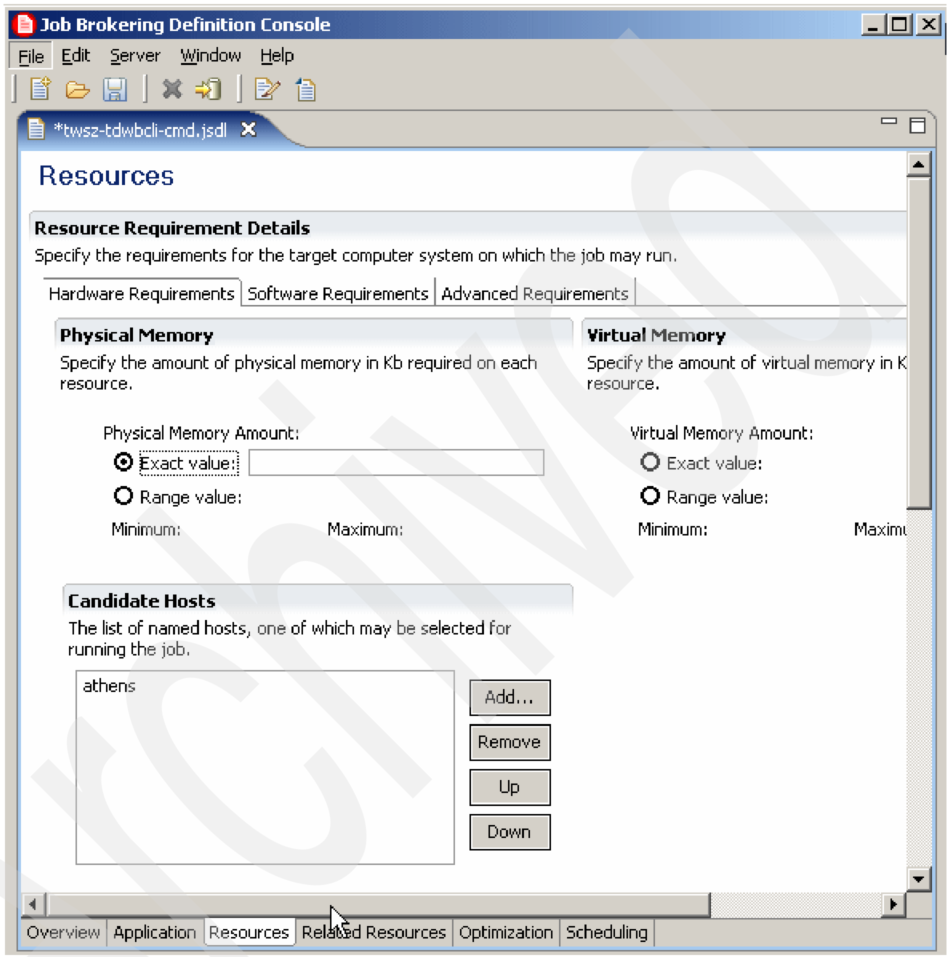
Value: ${tdwbcmd} ${tdwbcmdparms}

Figure 11-30 Executable and its parameters are added to Tivoli Dynamic Workload Broker job

Note: Credentials information was not supplied because the Tivoli Dynamic Workload Broker does not support it for Windows jobs. Instead, credentials information was supplied by customizing the CLIConfig.properties file located in server_installation_directory/config. Refer to the *Tivoli Dynamic Workload Broker User's Guide* for further details.

You must supply a user name and password when using a command-line interface if Tivoli Dynamic Workload Broker security is enabled, and this was the case in our scenario.

The last information needed in the job definition is the requirement that the job must run on the athens computer. This was done by adding the athens host name to the list of candidate hosts in the Resources pane, as shown in Figure 11-31.

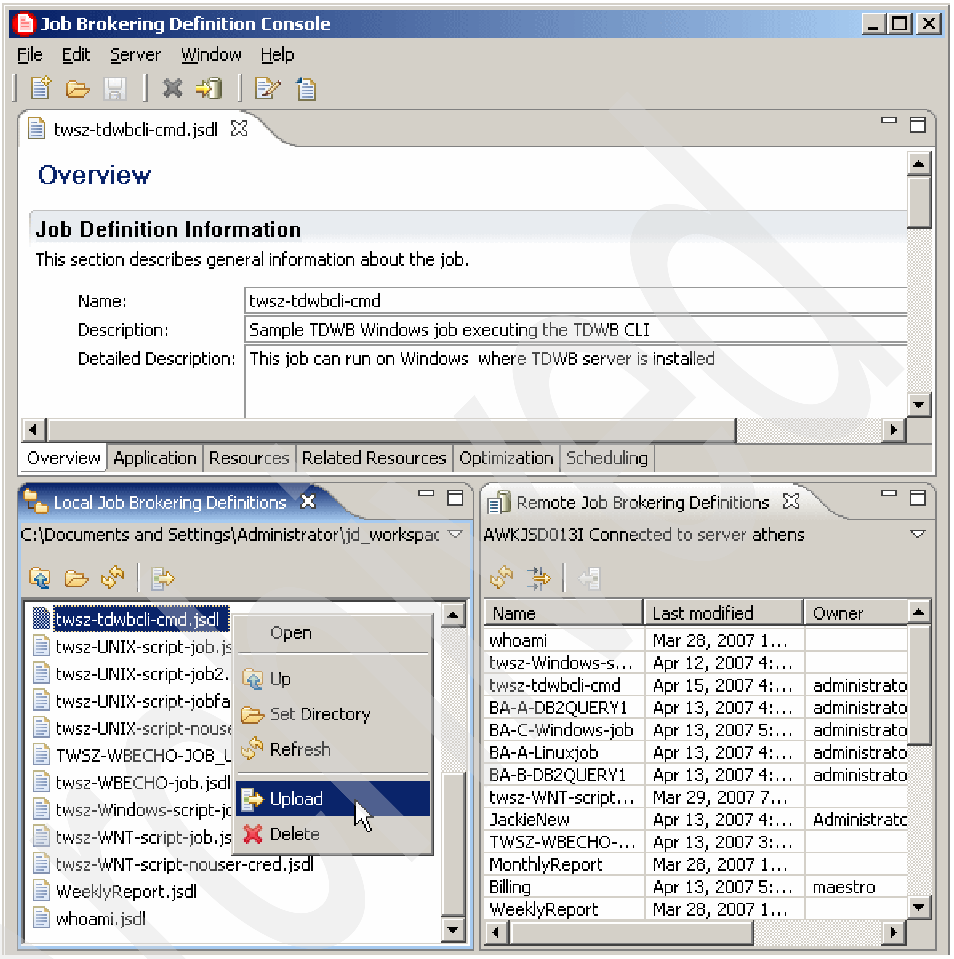


Physical Memory Amount: [✓ Exact value:] Range value:

Candidate Hosts: athens

Figure 11-31 Candidate host is added to Tivoli Dynamic Workload Broker job

Finally, the job definition was saved as twsz-tdwbcli-cmd.jsdl and uploaded to the Tivoli Dynamic Workload Broker job repository. This is shown in Figure 11-32.


Job Brokering Definition Console
File Edit Server Window Help
twsz-tdwbcli-cmd.jsdl
Overview

Job Definition Information

This section describes general information about the job.

Name:	twsz-tdwbcli-cmd
Description:	Sample TDWB Windows job executing the TDWB CLI
Detailed Description:	This job can run on Windows where TDWB server is installed

C:\Documents and Settings\Administrator\jd_workspac

twsz-tdwbcli-cmd.jsdl
twsz-UNIX-script-job.js
twsz-UNIX-script-job2.
twsz-UNIX-script-jobfa
twsz-UNIX-script-nouse
TWSZ-WBECHO-JOB_L
twsz-WBECHO-job.jsdl
twsz-Windows-script-jc
twsz-WNT-script-job.js
twsz-WNT-script-nouser-cred.jsdl
WeeklyReport.jsdl
whoami.jsdl

AWKJSD013I Connected to server athens

Name	Last modified	Owner
whoami	Mar 28, 2007 1...	
twsz-Windows-s...	Apr 12, 2007 4:...	
twsz-tdwbcli-cmd	Apr 15, 2007 4:...	administrato
BA-A-DB2QUERY1	Apr 13, 2007 4:...	administrato
BA-C-Windows-job	Apr 13, 2007 5:...	administrato
BA-A-Linuxjob	Apr 13, 2007 4:...	administrato
BA-B-DB2QUERY1	Apr 13, 2007 4:...	administrato
twsz-WNT-script...	Mar 29, 2007 7...	
JackieNew	Apr 13, 2007 4:...	Administratc
TWSZ-WBECHO-...	Apr 13, 2007 3:...	
MonthlyReport	Mar 28, 2007 1...	
Billing	Apr 13, 2007 5:...	maestro
WeeklyReport	Mar 28, 2007 1...	

Figure 11-32 Uploading job to TDWB job repository

Tivoli Workload Scheduler for z/OS end-to-end job with variables

A SCRPTPLIB member was created for usage with Tivoli Workload Scheduler for z/OS end-to-end job streams. The content of SCRPTLIB member WBCLI is shown in Example 11-20.

Example 11-20 SCRPTLIB member WBCLI

```
VARSUB
JOBREC
JOBCMD('twsz-tdwbcli-cmd -var tdwbcmdparms=-alias "TDWB#&OADID*"')
```

Let us look at the example in more detail:

- The VARSUB statement is used to instruct Tivoli Workload Scheduler for z/OS to perform variable substitution in the JOBREC statement.
- The JOBREC parameter JOBCMD() is used to specify two things. First, the name of the Tivoli Dynamic Workload Broker job to run (that is, twsz-tdwbcli-cmd), and second that the Tivoli Dynamic Workload Broker tdwbcmdparms job variable should be overridden with the value -alias "TDWB#&OADID*".
- The Tivoli Workload Scheduler for z/OS &OADID variable is used and it will be automatically substituted with the name of the job stream occurrence containing the job. Substitution occurs when the job is added to the current plan.

Note: The Tivoli Dynamic Workload Broker tdwbcmd job variable is not overridden in the end-to-end job definition, so it will take its defined default value "C:\Program Files\IBM\ITDWB\Server\bin\jobquery.bat" when the broker job twsz-tdwbcli-cmd is run.

The tdwbcmdparms job variable, however, is overridden as shown to make it possible to add the job to different job streams and get the desired results without having to edit the SCRPTLIB member each time. For the purpose of describing how this works, let us assume that job streams WBTDWBCLI and WBTDWBCLITEST are added to the current plan and that both job streams contain the job WBCLI. It works as follows:

1. Job stream WBTDWBCLI is added to the current plan. The &OADID variable in the JOBCMD() parameter is substituted with value WBTDWBCLI. The JOBCMD() value stored in the current plan and symphony file is:

   ```
   'twsz-tdwbcli-cmd -var tdwbcmdparms=-alias "TDWB#WBTDWBCLI*"'
   ```

2. Job stream WBTDWBCLITEST is added to the current plan. The &OADID variable in the JOBCMD() parameter is substituted with value WBTDWBCLITEST. The JOBCMD() value stored in the current plan and symphony file is:

   ```
   'twsz-tdwbcli-cmd -var tdwbcmdparms=-alias "TDWB#WBTDWBCLITEST*"'
   ```

3. Job WBCLI in job stream WBTDWBCLI is submitted. The Tivoli Dynamic Workload Broker runs job twsz-tdwbcli-cmd, which issues a CLI **`jobquery`** command listing all jobs with a job alias matching "TDWB#WBTDWBCLI*".
4. Job WBCLI in job stream WBTDWBCLITEST is submitted. The Tivoli Dynamic Workload Broker runs job twsz-tdwbcli-cmd, which issues a CLI **`jobquery`** command listing all jobs with a job alias matching "TDWB#WBTDWBCLITEST*".

Note: The Tivoli Workload Scheduler agent assigns a job alias to each job it forwards to the Tivoli Dynamic Workload Broker server.

In our lab environment the Tivoli Workload Scheduler agent received jobs from Tivoli Workload Scheduler for z/OS end-to-end V8.3. The format of the job alias assigned was:

```
<cpuschedname>#<schedname>.J<jobnumber>_<jobname>.SCHEDID-<schedid>.
ON-<yymmdd>.JNUM-<nnnn>
```

Where:

- *cpuschedname* is the job stream workstation.
- *schedname* is the job stream name.
- *jobname* is the job name.
- *schedid* is the job stream occurrence token.
- *yymmdd* is the schedule date of the job stream.
- *nnnn* is a job number created by the Tivoli Workload Scheduler agent.

The job alias can be used in command-line interface (for example, a jobquery):

```
jobquery.bat -alias "TDWB#WBTDWBCLITEST*"
```

That will return job information for jobs run on workstation TDWB, with a job stream name matching pattern WBTDWBCLITEST*.

Solution at work

To test the solution a Tivoli Workload Scheduler for z/OS end-to-end job stream WBTDWBCLITEST was created with three jobs, as shown in Table 11-4.

Table 11-4 Jobs in job stream WBTDWBCLITEST

workstation	Operation number	Job name	Extended job name
TDWB	005	WBQUERY	db2-query1
TDWB	010	WBQUERY	db2-query1
TDWB	050	WBCLI	twsz-tdwbcli-cmd

All three jobs are run at workstation TDWB, which is used for Tivoli Dynamic Workload Broker jobs. The extended job name is not required but was used to contain the Tivoli Dynamic Workload Broker job name.

The job stream runs two DB2 query jobs (job name WBQUERY). A third job WBCLI was added to submit the CLI query automatically after the two WBQUERY jobs have run. This job order is shown in Figure 11-33.

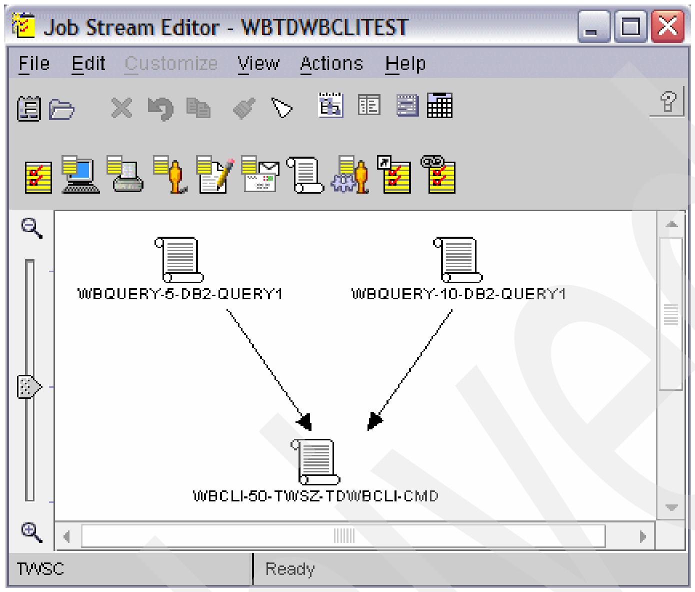


Figure 11-33 Job stream WBTDWBCLITEST in Job Scheduling Console job stream editor

The job stream was added to the current plan and all three jobs completed successfully.

Example 11-21 shows the WBCLI job log, which contains output from the **jobquery** command. The output has been slightly edited to enhance readability.

Example 11-21 WBCLI job log with output from TDWB CLI jobquery command

```
===============================================================
= JOB       : ATHENS#TWSZ-TDWBCLI-CMD
= USER      : TWSE2E
= JCLFILE   : TWSZ-TDWBCLI-CMD -VAR TD
= JOB NUMBER: 964470674
= MON APR 16 01:22:58 PDT 2007
===============================================================.
TWS AND TDWB ENVIRONMENT WAS SET UP SUCCESSFULLY!.

C:\>CALL "C:\PROGRAM FILES\IBM\ITDWB\SERVER\BIN\JOBQUERY.BAT"
-ALIAS "TDWB#WBTDW BCLITEST*"
```

```
CALL JOB DISPATCHER TO QUERY JOBS
SUCCESS RETURNED FROM JOB DISPATCHER
THERE ARE 3 JOBS FOUND FOR YOUR REQUEST
DETAILS ARE AS FOLLOWS:

JOB NAME: TWSZ-TDWBCLI-CMD.
JOB ALIAS: TDWB#WBTDWBCLITEST.J050_WBCLI.ON-20070316.JNUM-964470674.
JOB ID: 120A5F45-B86B-3A49-8D5C-09C55C3BCE0C.
JOB STATUS: EXECUTING.
JOB EPR: HTTP://ATHENS:9550/JDSERVICEWS/SERVICES/JOB.
JOB SUBMITTER: ADMINISTRATOR.
JOB SUBMITTER TYPE: N/A.
JOB SUBMIT TIME: MON APR 16 01:22:46 PDT 2007.
JOB START TIME: MON APR 16 01:22:58 PDT 2007.
JOB LAST STATUS MESSAGE: N/A.
JOB DURATION: N/A.
JOB RETURN CODE: 0.
JOB RESOURCE NAME: ATHENS.
JOB RESOURCE TYPE: COMPUTERSYSTEM.

JOB NAME: DB2-QUERY1.
JOB ALIAS: TDWB#WBTDWBCLITEST.J005_WBQUERY.ON-20070316.JNUM-964457596.
JOB ID: 251F5BBA-4F7A-3928-88AF-9CE11A301A27.
JOB STATUS: SUCCEEDED_EXECUTION.
JOB EPR:
HTTP://ATHENS:9550/JDSERVICEWS/.........................ERVICES/JOB.
JOB SUBMITTER: ADMINISTRATOR.
JOB SUBMITTER TYPE: N/A.
JOB SUBMIT TIME: MON APR 16 01:22:33 PDT 2007.
JOB START TIME: SUN APR 15 22:16:23 PDT 2007.
JOB END TIME: SUN APR 15 22:16:23 PDT 2007.
JOB LAST STATUS MESSAGE: N/A.
JOB DURATION: 0D 0H 0M 0S 0MS .
JOB RETURN CODE: 0.
JOB RESOURCE NAME: BARCELONA.ITSC.AUSTIN.IBM.COM.
JOB RESOURCE TYPE: COMPUTERSYSTEM.

JOB NAME: DB2-QUERY1.
JOB ALIAS: TDWB#WBTDWBCLITEST.J010_WBQUERY.ON-20070316.JNUM-964456924.
JOB ID: EA5A3383-CA66-3486-AA0E-D53C42569897.
JOB STATUS: SUCCEEDED_EXECUTION.
JOB EPR: HTTP://ATHENS:9550/JDSERVICEWS/SERVICES/JOB.
JOB SUBMITTER: ADMINISTRATOR.
JOB SUBMITTER TYPE: N/A.
JOB SUBMIT TIME: MON APR 16 01:22:33 PDT 2007.
JOB START TIME: SUN APR 15 22:16:22 PDT 2007.
JOB END TIME: SUN APR 15 22:16:22 PDT 2007.
JOB LAST STATUS MESSAGE: N/A.
JOB DURATION: 0D 0H 0M 0S 0MS .
JOB RETURN CODE: 0.
JOB RESOURCE NAME: BARCELONA.ITSC.AUSTIN.IBM.COM.
JOB RESOURCE TYPE: COMPUTERSYSTEM.
===============================================================.
= EXIT STATUS           : 0.
= SYSTEM TIME (SECONDS) : 3     ELAPSED TIME (MINUTES) : 0.
= USER TIME (SECONDS)   : 3.
= MON APR 16 01:23:01 PDT 2007.
===============================================================.
```

Note: When the WBCLI job is run the **jobquery** command is issued from TDWBCLI.cmd. The Tivoli Dynamic Workload Broker job twsz-tdwbcli-cmd will therefore appear in the jobquery output with job status EXECUTING.

11.6 Monitoring and control

Approaching and describing the monitoring and control of an integrated Tivoli Dynamic Workload Broker and Tivoli Workload Scheduler for z/OS end-to-end environment can be done in many ways.

We choose an approach where monitoring and control are addressed separately for two main areas that are further subdivided as follows:

- Monitoring and control of the infrastructure components that make job scheduling possible:
  - Tivoli Dynamic Workload Broker server, agent ,and Tivoli Workload Scheduler agent components
  - Tivoli Workload Scheduler for z/OS end-to-end engine, domain manager, and workstation components
  - Network infrastructure that enables interaction between components
- Monitoring and control of the workload being scheduled.
  - Monitoring workload progress
  - Modifying job streams and jobs in Tivoli Workload Scheduler for z/OS current plan
  - Job error handling

Our focus in this section is activities and considerations related to monitoring and control of workloads scheduled by Tivoli Workload Scheduler for z/OS end-to-end to Tivoli Dynamic Workload Broker.

For a more comprehensive coverage of the subject refer to:

- *IBM Tivoli Workload Scheduler for z/OS Managing the Workload,* SC32-1263
- *IBM Tivoli Workload Scheduler for z/OS Scheduling End-to-end* , SC32-1732
- *IBM Tivoli Dynamic Workload Broker User's Guide,* SC32-2281

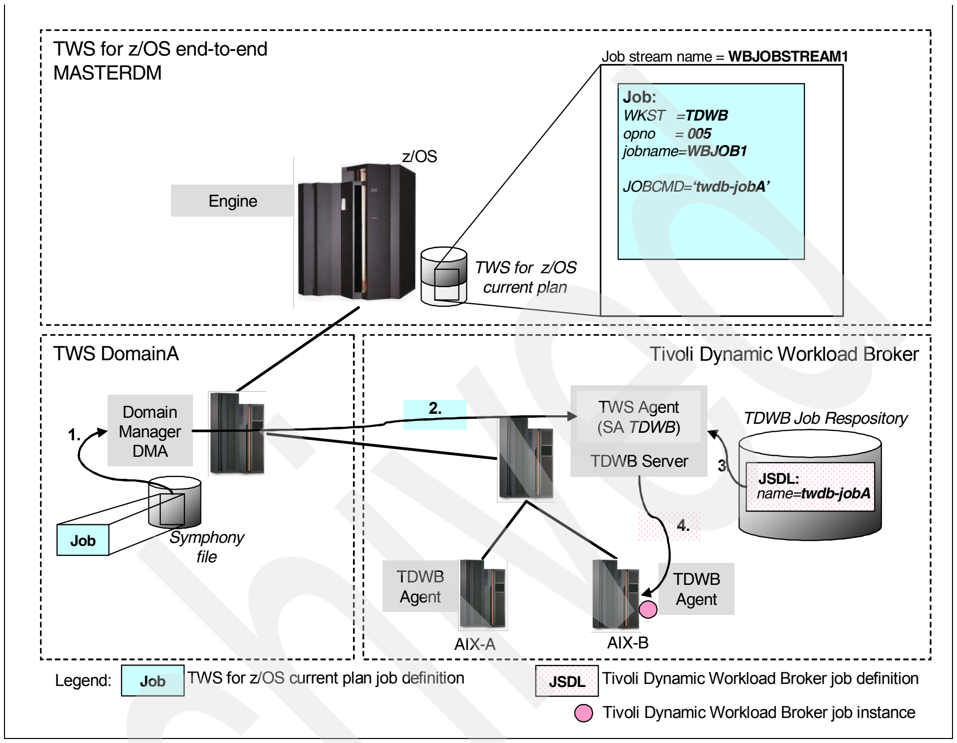
Note: In Tivoli Workload Scheduler for z/OS end-to-end the terms *job stream* and *application occurrence* are synonyms when talking about the workload content in the current plan. In this section we use both terms interchangeably.

11.6.1 Monitoring and control of infrastructure components

Outage or unavailability of one or more of the following infrastructure components can seriously degrade or prevent scheduling of Tivoli Dynamic Workload Broker workloads using Tivoli Workload Scheduler for z/S end-to-end:

- Tivoli Workload Scheduler for z/OS end-to-end components
  - Tivoli Workload Scheduler for z/OS engine
  - Tivoli Workload Scheduler domain managers
  - Tivoli Workload Scheduler for z/OS end-to-end workstation
- Tivoli Dynamic Workload Broker components
  - Tivoli Workload Scheduler agent
  - Tivoli Dynamic Workload Broker server
  - Tivoli Dynamic Workload Broker agents
- Network infrastructure that enables interaction between components

Figure 11-34 along with the description that follows show how the components interact when Tivoli Workload Scheduler for z/OS end-to-end submits a job to the Tivoli Dynamic Workload Broker.



TWS for z/OS end-to-end MASTERDM
Job stream name = WBJOBSTREAM1
Job:
WKST =TDWB
opno = 005
jobname=WBJOB1
JOBCMD='twdb-jobA'
z/OS
Engine
TWS for z/OS current plan
TWS DomainA
Tivoli Dynamic Workload Broker
1.
Domain Manager DMA
2.
TWS Agent (SA TDWB)
TDWB Server
TDWB Job Respository
3
JSDL: name=twdb-jobA
4.
Job
Symphony file
TDWB Agent
AIX-A
AIX-B
TDWB Agent
Legend:
Job
TWS for z/OS current plan job definition
JSDL
Tivoli Dynamic Workload Broker job definition
Tivoli Dynamic Workload Broker job instance

Figure 11-34 TWS for z/OS submitting job to Tivoli Dynamic Workload Broker

For the purpose of discussing Figure 11-34 we assume that job stream WBJOBSTREAM1 with job WBJOB1 has been defined to run on workstation TDWB. Workstation TDWB is used to submit jobs to the Tivoli Dynamic Workload Broker. The operation number of job WBJOB1 is 005. The WBJOB1 definition specifies that the Tivoli Dynamic Workload Broker job to run is tdwb-jobA.

The steps performed when the Tivoli Workload Scheduler for z/OS end-to-end submits job WBJOB1 to Tivoli Dynamic Workload Broker are:

1. Job stream WBJOBSTREAM1 is added to the current plan by the Tivoli Workload Scheduler for z/OS engine when:
   - The Daily Planning process creates the symphony file.
   - The job stream is added to the plan.

2. When job WBJOB1 is ready to run:

   Domain manager DMA:

   – (1) Retrieves the WBJOB1 definition from the symphony file. The information retrieved includes the name of the Tivoli Dynamic Workload Broker job definition to be launched (tdwb-jobA in our case).
   – (2) Sends a job launch request to the Tivoli Workload Scheduler agent, which in turn transfers the job launch request to the Tivoli Dynamic Workload Broker server.

   The Tivoli Dynamic Workload Broker server:

   – (3) Retrieves the JSDL definition with name tdwb-jobA.
   – Allocates resources to the job, including a target computer. (Logical and physical resource requirements for tdwb-jobA as well as current resource usage data is used to find a best-fit resource.)
   – (4) Submits a job request to its agent on the AIX machine selected.
   – Waits for the execution result and when it arrives forwards it to the Tivoli Workload Scheduler agent, which in turn forwards it to domain manager DMA.

3. Domain manager DMA

   – Updates its symphony file. Job stream and job status are updated to reflect execution result.
   – Propagates execution result to the Tivoli Workload Scheduler for z/OS engine.

4. Tivoli Workload Scheduler for z/OS engine updates the current plan. Job stream and job status are updated to reflect the execution result.

It should be obvious from this walk-through that monitoring and control of the infrastructure components are essential. Monitoring and control can be performed manually by people, automatically using software such as IBM Tivoli Enterprise Portal, or both. What to describe and what to consider are therefore very much dependent on the actual installation implementation and organization.

Some considerations on monitoring and control of the Tivoli Workload Scheduler for z/OS end-to-end workstation will be given in the next subtopic, but for further coverage of the subject we refer the reader to the product manauls.

Monitoring and control of workstation

When monitoring and controlling the Tivoli Workload Scheduler for z/OS end-to-end workstation that represents the Tivoli Workload Scheduler agent keep in mind the following considerations:

- The Tivoli Workload Scheduler for z/OS end-to-end workstation used for scheduling Tivoli Dynamic Workload Broker jobs is an emulated Tivoli Workload Scheduler standard agent.
  - The workstation is emulated by the Tivoli Workload Scheduler agent, which is a component of the Tivoli Dynamic Workload Broker. The Tivoli Workload Scheduler agent is a WebSphere Enterprise Application and it runs on the same WebSphere as the Tivoli Dynamic Workload Broker TWS server component.
  - The workstation is started by issuing a link command.
  - The workstation is stopped by issuing an unlink command.
  - The workstation commands start and stop are not supported.
- The Tivoli Workload Scheduler agent component is automatically started when the broker server is started and an emulated Netman function begins listening for requests from its hosting Tivoli Workload Scheduler domain manager.

Note: Starting and stopping the Tivoli Workload Scheduler agent workstation does not follow the normal rules for a Tivoli Workload Scheduler standard agent workstation. For the Tivoli Workload Scheduler agent workstation the following applies:

- The link command both links and starts the workstation.
- The unlink command both stops and unlinks the workstation.

When you do a successful link and unlink of a Tivoli Workload Scheduler agent workstation this is logged both in the Tivoli Workload Scheduler for z/OS end-to-end logs and in the Tivoli Dynamic Workload Broker log.

Example 11-22 shows sample Tivoli Workload Scheduler for z/OS end-to-end messages from our lab environment.

Example 11-22 TWS for z/OS end-to-end log messages resulting from link and unlink of workstation

```
Messages in TWS for z/OS Controller mlog when issuing Link command.
EQQWL50I COMMAND LINK SENT FOR TDWB
EQQWL10W workstation TDWB HAS BEEN SET TO LINKED STATUS TYPE SAGENT DOMAIN MASTERDM
EQQWL10W workstation TDWB HAS BEEN SET TO ACTIVE STATUS TYPE SAGENT DOMAIN MASTERDM

Messages in TWS for z/OS USS server TWSMERGE.log when issuing Link command.
```

```
MAILMAN:AWSBCV018I ÆJ67175125/Operator command: LINK TDWB
MAILMAN:AWSBCV029I Attempting to link to TDWB.
MAILMAN:AWSBCV056I Mailman has tried to link to a workstation (TDWB) to which it is
already linked.
MAILMAN:AWSBCV104I Has linked to TDWB using TCP.
BATCHMAN:AWSBDY104I Received command MY:LINK for run number 32 for workstation TDWB
from workstation OPCMASTER.T
BATCHMAN:Workstation TDWB State is being changed: LINKED=TCP
BATCHMAN:AWSBHT033I Workstation TDWB is now active, scheduling is resuming.
TDWB:WRITER:AWSBCW028I Started by MAILMAN/8.3 from TDWB; workstation type: WNT
TDWB:WRITER:AWSBCW031I Handshake command_type StartMailbox
BATCHMAN:AWSBDY112I Received command MY:WRITER-UP for run number -1 for workstation
TDWB from workstation OPCMASTER.
BATCHMAN:Workstation TDWB State is being changed: WRITER
BATCHMAN:AWSBDY110I Received command MY:JOBMAN-UP for run number 32 from workstation
TDWB.
BATCHMAN:Workstation TDWB State is being changed: JOBMAN

Messages in TWS for z/OS Controler mlog when issuing Unlink command.
EQQWL50I COMMAND UNLINK SENT FOR TDWB
EQQWL10W workstation TDWB HAS BEEN SET TO UNLINKED STATUS TYPE SAGENT DOMAIN MASTERDM
EQQWL10W workstation TDWB HAS BEEN SET TO OFFLINE STATUS TYPE SAGENT DOMAIN MASTERDM

Messages in TWS for z/OS USS server TWSMERGE.log when issuing Unlink command.
MAILMAN:AWSBCV018I ÆJ67175125/Operator command: UNLINK TDWB
MAILMAN:+
MAILMAN:+ AWSBCV027I Unlinking from TDWB
MAILMAN:+
MAILMAN:AWSBCV028I Unlinked from TDWB, and will write to the PO box.
BATCHMAN:AWSBDY103I Received command MY:UNLINK for run number 32 for workstation
OPCMASTER
BATCHMAN:Workstation TDWB State is being changed: UNSETTING: LINKED=TCP
BATCHMAN:AWSBHT032I Workstation TDWB is now inactive, no jobs will be scheduled.
```

Example 11-23 shows sample Tivoli Dynamic Workload Broker messages from our lab environment.

Example 11-23 Tivoli Dynamic Workload Broker log messages resulting from link and unlink of workstation

```
Messages in Tivoli Dynamic Workload Broker Systemout.log when issuing Link command.
TWSAgent      I   AWKTSA010I The START WRITER service request has been received by
Netman.
TWSAgent      I   AWKTSA017I Writer has been successfully started and it is listening
for messages.
```

```
TWSAgent      I   AWKTSA032I The TWS Agent CPU has been successfully linked to
[Ljava.lang.Object;@2ab7915e using port {1}.
TWSAgent      I   AWKTSA026I Jobman has been successfully started.
TWSAgent      I   AWKTSA021I Mailman has been successfully started and the uplink
connection is established.

Messages in Tivoli Dynamic Workload Broker Systemout.log when issuing Unlink command.
TWSAgent      I   AWKTSA028I Jobman has been successfully stopped.
TWSAgent      I   AWKTSA036I The TWS Agent CPU has been successfully unlinked.
TWSAgent      I   AWKTSA024I Mailman has been successfully stopped.
```

11.6.2 Monitoring and control of the workload being scheduled

Monitoring and control of the workload being scheduled are the focus of this section, and more specifically we describe considerations that apply when Tivoli Workload Scheduler for z/OS end-to-end is used to run Tivoli Dynamic Workload Broker workloads.

In a Tivoli Workload Scheduler for z/OS end-to-end environment your business applications and tasks are mapped by scheduling analysts into units of work called jobs. Jobs are grouped into job streams along with times, priorities, and other dependencies that determine the exact order of the jobs. Job streams also contain business run schedules. Based on this information the Tivoli Workload Scheduler for z/OS end-to-end regularly creates the current plan, which contains the workload for both the mainframe and end-to-end scheduling network.

We describe the following areas:

- Monitoring workload progress, both from a human perspective and by automatic job notification mechanisms
- Modifying job streams and jobs in the Tivoli Workload Scheduler for z/OS current plan
- Job error handling

Note: In this section we show user interface samples to illustrate the look-and-feel of available user interfaces. Detailed step-by-step user interface actions have been deliberately left out.

Monitoring workload progress

From a human perspective monitoring workload progress almost intuitively brings the concept of state, or status, into focus.

Tivoli Workload Scheduler for z/OS end-to-end job status

Tivoli Workload Scheduler for z/OS end-to-end assigns a *status code* to every job stream and every job in the current plan. An *error code* is also assigned for any job that ends in error. Tivoli Workload Scheduler for z/OS end-to-end also maintains an *extended status code* that provides additional information about the status of a job. However, the extended status code is not always present.

You monitor (that is, visually display) the status of the job stream and jobs by using the Tivoli Workload Scheduler for z/OS end-to-end user interfaces:

- Job Scheduling Console (JSC)
- Tivoli Workload Scheduler for z/OS ISPF panels
- Tivoli Dynamic Workload Console

We decided to show examples using JSC and ISPF panels.

The *job status codes* relevant for jobs submitted to the Tivoli Dynamic Workload Broker are:

W	The job is waiting for a predecessor to complete.
A	Arriving. The job is ready for processing. No predecessors were defined.
R	Ready for processing. All predecessors are complete.
*****	Ready. At least one predecessor is defined on a non reporting. workstation. All predecessors are complete.
S	Started.
C	Complete.
E	The job has ended in error.
D	Deleted.

Tivoli Dynamic Workload Broker job status

The Tivoli Dynamic Workload Broker server assigns a *job status* to every job instance.

In a native Tivoli Dynamic Workload Broker environment you monitor ()hat is, visually display, the job status by using the Tivoli Dynamic Workload Broker user interfaces:

- Tivoli Dynamic Workload Broker Web Console
- Tivoli Dynamic Workload Broker command-line interface

In an integrated environment you can also monitor Tivoli Dynamic Workload Broker job status using the status code assigned by the Tivoli Workload Scheduler for z/OS end-to-end to the corresponding job in the current plan. How to map job status between the products is our next subtopic.

Status mapping

Table 11-5 shows how to map between:

- Tivoli Dynamic Workload Broker job status
- Tivoli Workload Scheduler for z/OS end-to-end job status code and extended status code
- Tivoli Workload Scheduler Job Scheduling Console job status

Table 11-5 Status mapping between Tivoli Dynamic Workload Broker and TWS for z/OS end-to-end

Tivoli Dynamic Workload Broker job status	**Tivoli Workload Scheduler for z/OS end-to-end job status code and extended status code**	**Tivoli Workload Scheduler Job Scheduling Console job status**
1) Not applicable	W - Waiting	Waiting
	A - Arriving	Ready
	R - Ready	Ready
	* - Ready	Ready
	2) E - Error Error code is set to FAIL.	2) Error Error code is set to FAIL.
► Submitted ► Waiting for resources ► Waiting for reallocation ► Resource allocation received ► Submitted to agent	3) S - Started Extended status code is set to Q.	3) Running Status details: Added job to TDWB job queue.
► Running	S - Started Extended status code is set to S.	Running
► Completed successfully	C - Completed	Successful
4) ► Run Failed	4) E - Error Error code is set to nnnn.	4) Error Error code is set to nnnn.
► Resource allocation failed ► Unable to start	E - Error Error code is set to FAIL.	Error Error code is set to FAIL.

Tivoli Dynamic Workload Broker job status	Tivoli Workload Scheduler for z/OS end-to-end job status code and extended status code	Tivoli Workload Scheduler Job Scheduling Console job status
► Canceled	E - Error Error code is set to 0000.	Error Error code is set to 0000.
► Cancel pending ► Cancel allocation	The status code will be set to: ► E - Error with error code set to 0000 when the job reaches Canceled state in IBM Tivoli Dynamic Workload Broker.	The job status will be set to: ► E - Error with error code set to 0000 when the job reaches Canceled state in IBM Tivoli Dynamic Workload Broker.

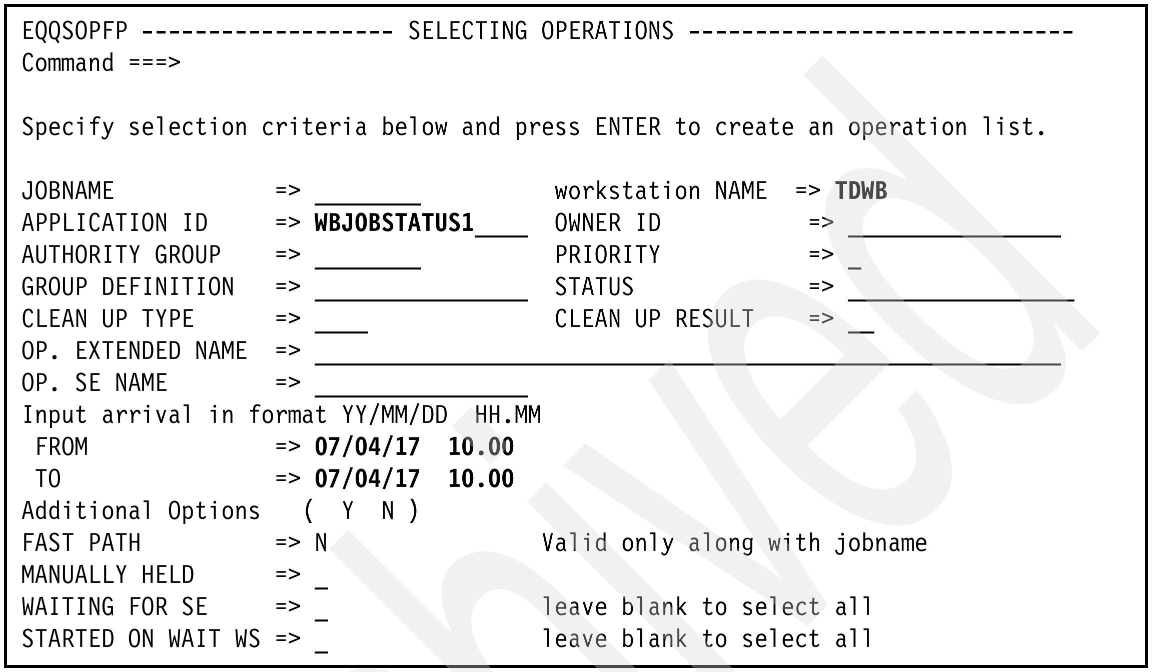
Notes:
1) Before the Tivoli Dynamic Workload Broker server has received a jobsubmit request from the Tivoli Workload Scheduler agent there is no job instance and hence there is no applicable Tivoli Dynamic Workload Broker job status.
2) Internal processing errors can occur in the Tivoli Workload Scheduler agent that prevent it from processing an incoming job request from its hosting Tivoli Workload Scheduler domain manager. In this case the job will fail with error code=FAIL. An example is when the Tivoli Workload Scheduler agent fails authentication itself to the Tivoli Dynamic Workload Broker server.
3) The extended status is S, which is a program defect.
4) The Tivoli Dynamic Workload Broker job status *run failed* is set when job execution failed (that is, a Tivoli Dynamic Workload Broker agent has submitted the job to an operating system but execution failed for some reason). The Tivoli Workload Scheduler for z/OS end-to-end job status code is set to *E- Error* along with a numerical error code.

Job status display using z/OS ISPF panels

Let us first see how Tivoli Workload Scheduler for z/OS ISPF panels show job status for:

- Job stream occurrence WBJOBSTATUS1
- Jobs running on workstation TDWB

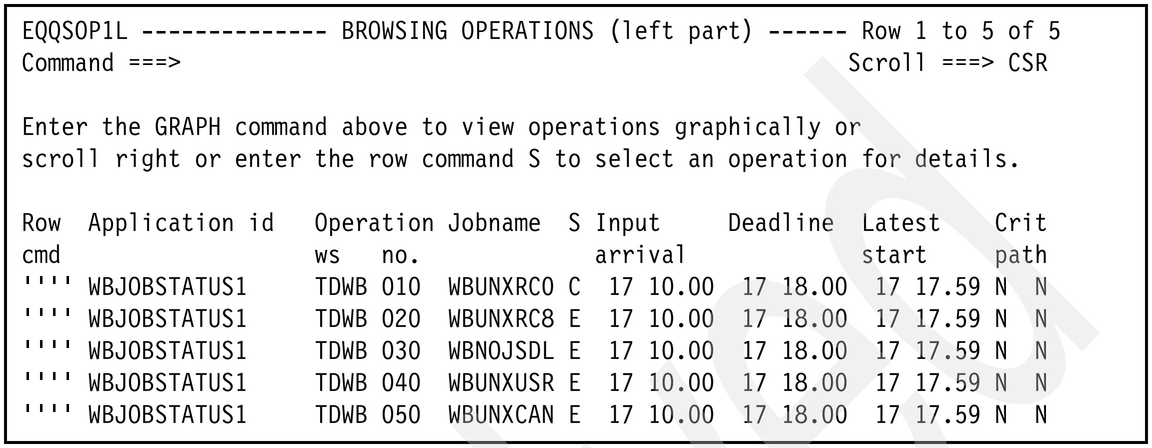
We generated a display showing jobs for a single job stream occurrence in the current plan. To achieve this we used filter criteria on Tivoli Workload Scheduler for z/OS ISPF panel EQQSOPFP, as shown with bold text in Figure 11-35.



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

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

JOBNAME            => ________          workstation NAME  => TDWB
APPLICATION ID     => WBJOBSTATUS1____  OWNER ID           => ________________
AUTHORITY GROUP    => ________          PRIORITY           => _
GROUP DEFINITION   => ________________  STATUS             => _________________
CLEAN UP TYPE      => ____              CLEAN UP RESULT    => __
OP. EXTENDED NAME  => ________________________________________________________
OP. SE NAME        => ________________
Input arrival in format YY/MM/DD  HH.MM
 FROM              => 07/04/17  10.00
 TO                => 07/04/17  10.00
Additional Options   (  Y  N )
FAST PATH          => N                Valid only along with jobname
MANUALLY HELD      => _
WAITING FOR SE     => _                leave blank to select all
STARTED ON WAIT WS => _                leave blank to select all
```

Figure 11-35 Entering job display criteria on TWS for z/OS ISPF panel EQQSOPFP

Figure 11-36 shows the resulting panel EQQSOP1L displaying job status information in column S. There are four jobs in error status and one job in complete status.



```
EQQSOP1L -------------- BROWSING OPERATIONS (left part) ------ Row 1 to 5 of 5
Command ===>                                                  Scroll ===> CSR

Enter the GRAPH command above to view operations graphically or
scroll right or enter the row command S to select an operation for details.

Row  Application id   Operation Jobname  S Input     Deadline  Latest    Crit
cmd                   ws   no.             arrival             start     path
'''' WBJOBSTATUS1     TDWB 010  WBUNXRC0 C  17 10.00  17 18.00  17 17.59 N  N
'''' WBJOBSTATUS1     TDWB 020  WBUNXRC8 E  17 10.00  17 18.00  17 17.59 N  N
'''' WBJOBSTATUS1     TDWB 030  WBNOJSDL E  17 10.00  17 18.00  17 17.59 N  N
'''' WBJOBSTATUS1     TDWB 040  WBUNXUSR E  17 10.00  17 18.00  17 17.59 N  N
'''' WBJOBSTATUS1     TDWB 050  WBUNXCAN E  17 10.00  17 18.00  17 17.59 N  N
```

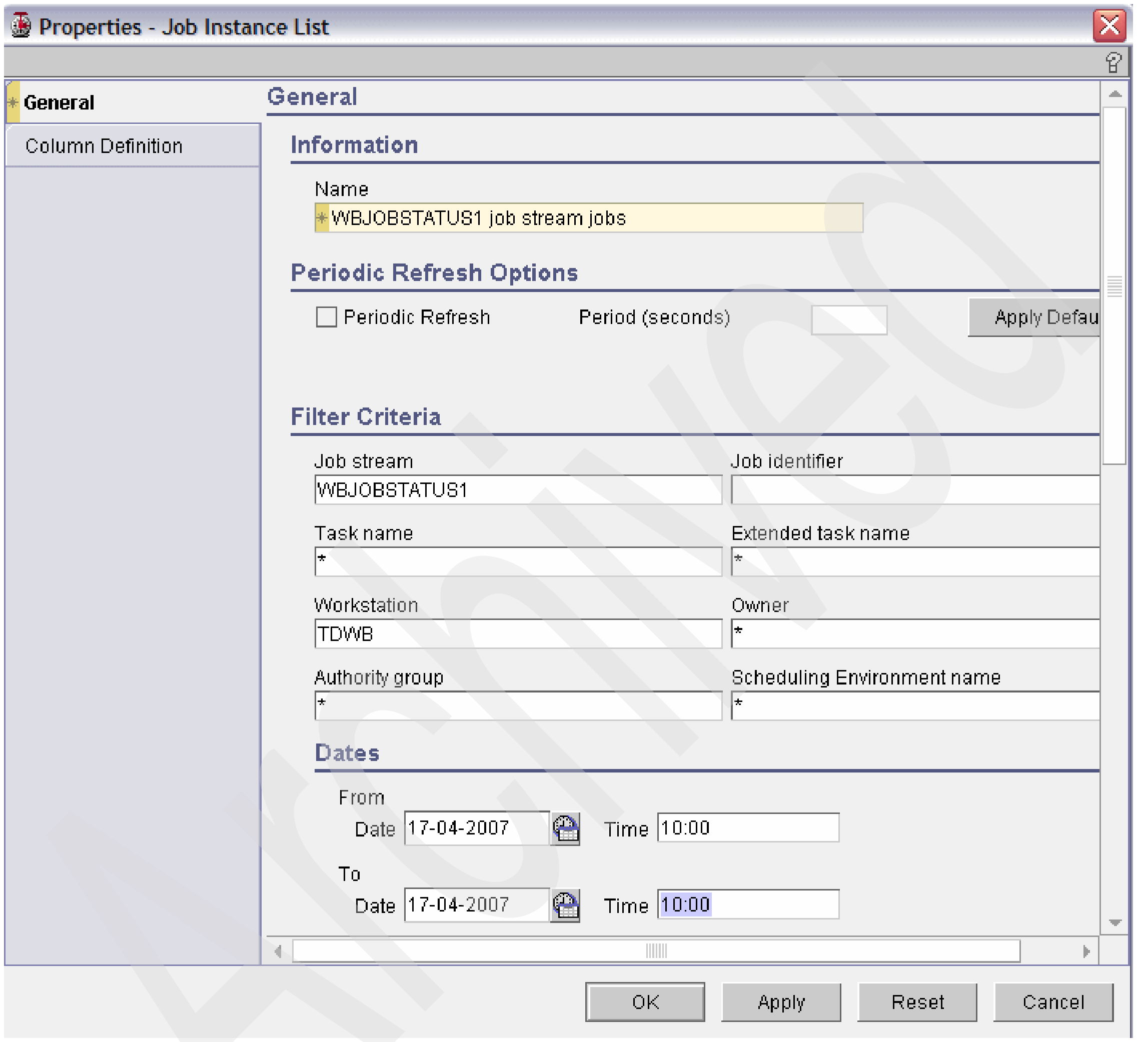
Figure 11-36   Job status display on TWS for z/OS ISPF panel EQQSOP1L

Job status display using Job Scheduling Console

Now let us see how the job status is displayed using the Job Scheduling Console for:

- Job stream occurrence WBJOBSTATUS1
- Jobs running on workstation TDWB

First we generated a display showing jobs for a single job stream occurrence in the current plan. To achieve this we created a Job Scheduling Console job instance list with filter criteria, as shown in Figure 11-37.


Properties - Job Instance List

General
Column Definition

General

Information

Name
WBJOBSTATUS1 job stream jobs

Periodic Refresh Options

Periodic Refresh   Period (seconds)   Apply Defau

Filter Criteria

Job stream: WBJOBSTATUS1	Job identifier:
Task name: *	Extended task name: *
Workstation: TDWB	Owner: *
Authority group: *	Scheduling Environment name: *

Dates

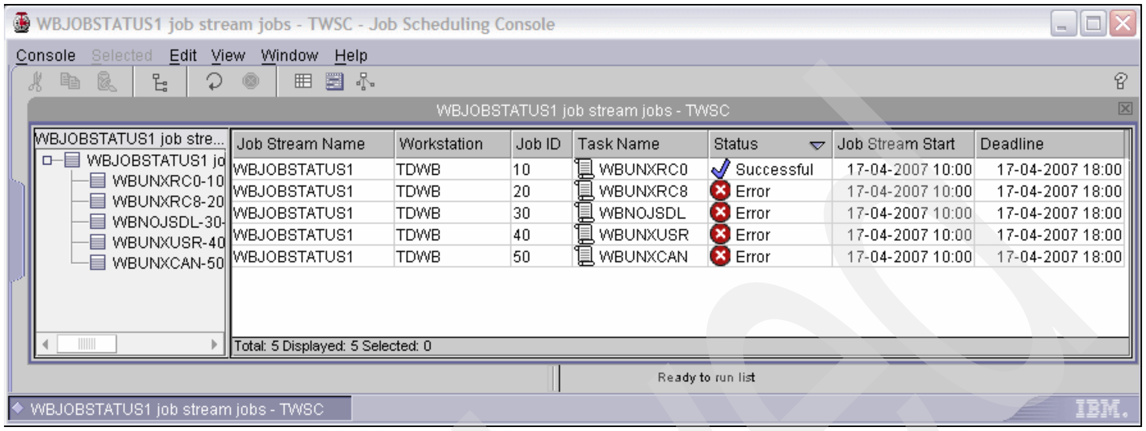
From
Date 17-04-2007 Time 10:00

To
Date 17-04-2007 Time 10:00

OK Apply Reset Cancel

Figure 11-37   Defining job display criteria on Job Scheduling Console job instance list

Figure 11-38 shows the resulting JSC window displaying job status information in the Status column. There are four jobs in error status and one job in successful status.


WBJOBSTATUS1 job stream jobs - TWSC - Job Scheduling Console
Console Selected Edit View Window Help
WBJOBSTATUS1 job stream jobs - TWSC
WBJOBSTATUS1 job stre...
WBJOBSTATUS1 jo
WBUNXRC0-10
WBUNXRC8-20
WBNOJSDL-30-
WBUNXUSR-40
WBUNXCAN-50
Ready to run list
WBJOBSTATUS1 job stream jobs - TWSC
IBM.

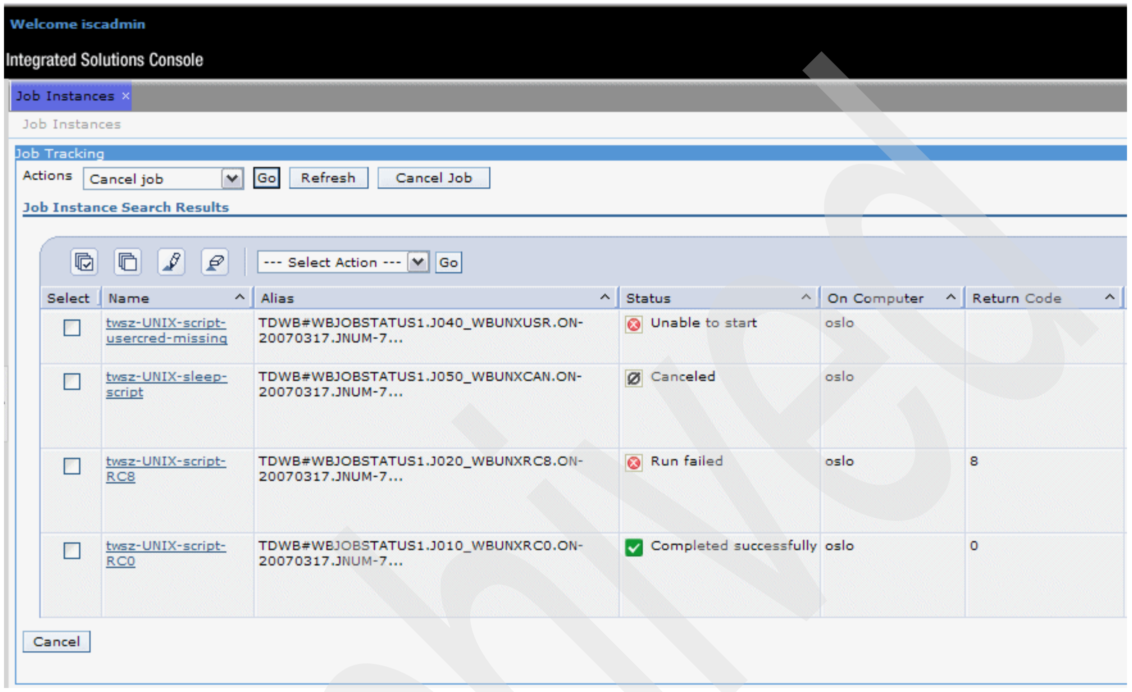
Job Stream Name	Workstation	Job ID	Task Name	Status	Job Stream Start	Deadline
WBJOBSTATUS1	TDWB	10	WBUNXRC0	✓ Successful	17-04-2007 10:00	17-04-2007 18:00
WBJOBSTATUS1	TDWB	20	WBUNXRC8	✗ Error	17-04-2007 10:00	17-04-2007 18:00
WBJOBSTATUS1	TDWB	30	WBNOJSDL	✗ Error	17-04-2007 10:00	17-04-2007 18:00
WBJOBSTATUS1	TDWB	40	WBUNXUSR	✗ Error	17-04-2007 10:00	17-04-2007 18:00
WBJOBSTATUS1	TDWB	50	WBUNXCAN	✗ Error	17-04-2007 10:00	17-04-2007 18:00

Total: 5 Displayed: 5 Selected: 0

Figure 11-38 Job status display using Job Scheduling Console

Job status display using Tivoli Dynamic Workload Broker

Now let us see how job status is displayed using the Tivoli Dynamic Workload Broker Web Console. Figure 11-39 shows the job status for four job instances.



Select	Name	Alias	Status	On Computer	Return Code
☐	twsz-UNIX-script-usercred-missing	TDWB#WBJOBSTATUS1.J040_WBUNXUSR.ON-20070317.JNUM-7...	Unable to start	oslo	
☐	twsz-UNIX-sleep-script	TDWB#WBJOBSTATUS1.J050_WBUNXCAN.ON-20070317.JNUM-7...	Canceled	oslo	
☐	twsz-UNIX-script-RC8	TDWB#WBJOBSTATUS1.J020_WBUNXRC8.ON-20070317.JNUM-7...	Run failed	oslo	8
☐	twsz-UNIX-script-RC0	TDWB#WBJOBSTATUS1.J010_WBUNXRC0.ON-20070317.JNUM-7...	✓ Completed successfully	oslo	0

Figure 11-39 Job status display using Tivoli Dynamic Workload Broker Web Console

Let us take a close look at Tivoli Dynamic Workload Broker job instance information in Figure 11-39 and compare this to the Tivoli Workload Scheduler for z/OS end-to-end job information shown on the Job Schdeulig Console in Figure 11-38 on page 615.

Table 11-6 shows how information is related.

Table 11-6 Job Scheduling Console and Web Console job status

JSC job name	JSC job status	Web Console job name	Web Console job status
WBUNXRC0	Successful	twsz-UNIX-script-RC0	Completed successfully
WBUNXRC8	Error	twsz-UNIX-script-RC8	Run failed
WBNOJSDL	Error		
WBUNXUSR	Error	twsz-UNIX-script-userc red-missing	Unable to start
WBUNXCAN	Error	twsz-UNIX-sleep-script	Canceled

How did we correlate the jobs to each other? We used the Tivoli Dynamic Workload Broker job alias. In our case the Tivoli Dynamic Workload Broker job alias is in the format:

```
TDWB#WBJOBSTATUS1.J<jobnumber>_<jobname>...
```

The job number and job name of the job alias were used to identify the corresponding job on the Job Scheduling Console.

Note: Job WBNOJSDL in the Job Scheduling Console list has no corresponding job in the Web Console job instance list. This is because we tried to run Tivoli Dynamic Workload Broker job JSDL that did not exist in the Tivoli Dynamic Workload Broker job respository. Hence, no Tivoli Dynamic Workload Broker job instance could be created.

Automatic notification about job conditions

Automatic notification about job conditions is a way to relieve operations analysts from the burden of manually monitoring workload progress of perhaps hundreds of thousands of jobs. The idea is to selectively create notifications only for job conditions that inform of actual or possible impact on workload progress.

Automatic notification of job conditions can come from Tivoli Workload Scheduler for z/OS and from the Tivoli Dynamic Workload Broker.

Tivoli Workload Scheduler for z/OS job notification

Tivoli Workload Scheduler for z/OS provides the following job notifications:

- Job long duration

  The job long duration alert action is taken when a job in the current plan is active for an unexpectedly long time.

- Job error

  The job error alert action is taken when a job in the current plan is set to ended-in-error status.

- Job late

  The job late alert action is taken when a job in the current plan becomes late. A job is considered late if it reaches its latest start time and does not have the status started, complete, or deleted.

IBM Tivoli Workload Scheduler for z/OS controller can take several actions when a job notification alert is generated. Available actions are:

- Write a message to the Tivoli Workload Scheduler for z/OS controller message log.
- Write a write-to-operator (WTO) message to the z/OS system log.
- Send a generic alert to NetView®.
- Send a generic alert to IBM Tivoli Monitoring agent.

  Using the Tivoli Monitoring agent, Tivoli Workload Scheduler for z/OS can work with Tivoli Business Systems Manager and Tivoli Monitoring through the Tivoli Enterprise Portal component to provide external monitoring of job notifications.

Tivoli Dynamic Workload Broker job notification

Tivoli Dynamic Workload Broker job notifications are described in 4.6, "Monitoring computers and jobs" on page 171.

Modifying job streams and jobs in current plan

Job streams and jobs that contains workloads destined for Tivoli Dynamic Workload Broker can be managed in real time using the advanced capabilities of Tivoli Workload Scheduler for z/OS end-to-end, thereby further delivering integration benefits.

Using Tivoli Workload Scheduler for z/OS end-to-end you can manage objects in the current plan:

- Job streams

  You can:

  - View the properties.
  - View successors and predecessors.
  - Rerun a job stream (all jobs or selected jobs).
  - Complete and delete the job stream.
  - Add job streams ad hoc to the current plan (automatically resolving predecessors and successors as needed).

- Jobs

  You can:

  - View the properties.
  - View and manage successors and predecessors.
  - Rerun, complete, and delete jobs.
  - Manage job recovery.
  - Browse the job log.
  - Stop a running job.

- Tivoli Workload Scheduler workstation

  You can:

  - View the properties.
  - Change the job limit.
  - Link or unlink.

Managing objects in the current plan are documented in:

- *IBM Tivoli Workload Scheduler for z/OS Managing the Workload*, SC32-1263
- *IBM Tivoli Workload Scheduler for z/OS Scheduling End-to-end,* SC32-1732

We now describe considerations that apply when managing Tivoli Dynamic Workload Broker jobs:

- Stopping a running job
- Job error handling

Stopping a running Tivoli Dynamic Workload Broker job

We define a running Tivoli Dynamic Workload Broker job as a job for which all of the following statements are true:

- ► The corresponding Tivoli Workload Scheduler for z/OS end-to-end job is in status S - Started.
- ► The job has been submitted to an operating system by a Tivoli Dynamic Workload Broker agent.
- ► The job is currently executing.

You can stop (that is, cancel) job execution in several ways:

- ► Initiate a kill action from Tivoli Workload Scheduler for z/OS ISPF panels.
- ► Initiate a kill action from the Job Scheduling Console.
- ► Initiate a cancel action from the Tivoli Dynamic Workload Broker Web Console.
- ► Initiate a cancel action using the Tivoli Dynamic Workload Broker CLI command **`jobcancel`**.
- ► Cancel job execution on the computer where it is currently running.

Kill actions initiated from Tivoli Workload Scheduler for z/OS result in a job cancel request being sent to the Tivoli Dynamic Workload Broker server.

Cancel actions initiated from the Tivoli Dynamic Workload Broker Web Console or CLI result in a job cancel request being sent to the Tivoli Dynamic Workload Broker server.

The Tivoli Dynamic Workload Broker server forwards an incoming job cancel request to the Tivoli Dynamic Workload Broker agent on the computer were the job is currently executing.

The Tivoli Dynamic Workload Broker agent asks the operating system to cancel job execution. The Tivoli Dynamic Workload Broker agent captures the job termination result and reports it back to its server for further handling.

Note: The following Tivoli Workload Scheduler for z/OS object actions *do not* initiate any job level kill or cancel actions:

- ► Complete, rerun, or delete job stream.
- ► Complete, rerun, or delete job.
- ► Unlink workstation.

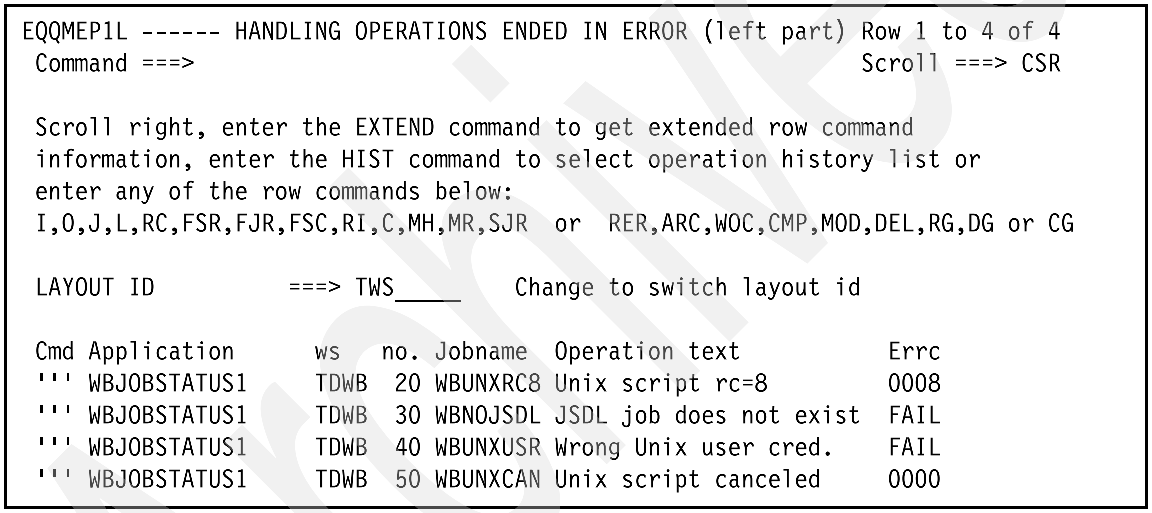
Job error handling

Manual error handling of Tivoli Dynamic Workload Broker jobs submitted from the Tivoli Workload Scheduler for z/OS end-to-end is managed using:

- Tivoli Workload Scheduler for z/OS ISPF panels
- Job Scheduling Console

Tivoli Workload Scheduler for z/OS end-to-end also delivers capabilities to automate job error handling. This is called job recovery and is described in 11.5.5, “Job recovery and restart” on page 582.

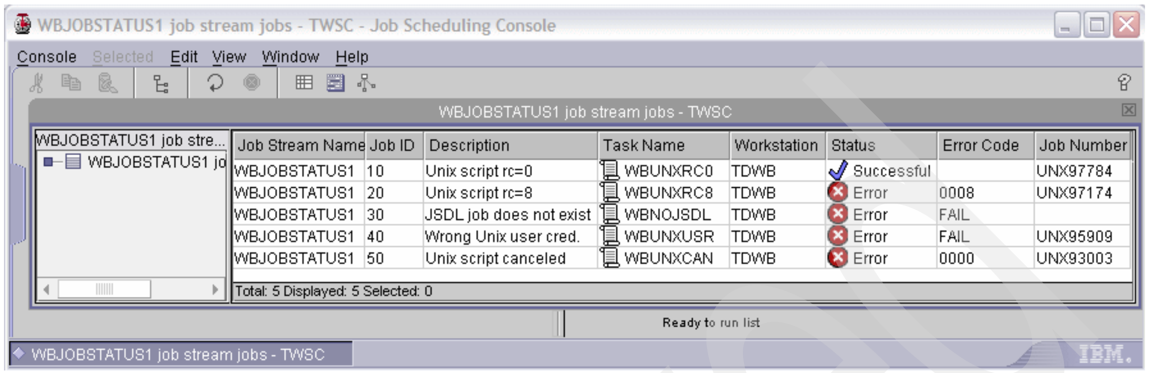
Job error handling using Tivoli Workload Scheduler for z/OS ISPF panels should preferrably be perfomed from the error list panel. Figure 11-40 shows an example of this panel with four jobs in error status. The job error codes are shown in column Errc.



```
EQQMEP1L ------ HANDLING OPERATIONS ENDED IN ERROR (left part) Row 1 to 4 of 4
 Command ===>                                                 Scroll ===> CSR

 Scroll right, enter the EXTEND command to get extended row command
 information, enter the HIST command to select operation history list or
 enter any of the row commands below:
 I,O,J,L,RC,FSR,FJR,FSC,RI,C,MH,MR,SJR  or  RER,ARC,WOC,CMP,MOD,DEL,RG,DG or CG

 LAYOUT ID          ===> TWS_____    Change to switch layout id

 Cmd Application      ws   no. Jobname  Operation text          Errc
 ''' WBJOBSTATUS1     TDWB  20 WBUNXRC8 Unix script rc=8        0008
 ''' WBJOBSTATUS1     TDWB  30 WBNOJSDL JSDL job does not exist  FAIL
 ''' WBJOBSTATUS1     TDWB  40 WBUNXUSR Wrong Unix user cred.    FAIL
 ''' WBJOBSTATUS1     TDWB  50 WBUNXCAN Unix script canceled     0000
```

Figure 11-40 Tivoli Workload Scheduler for z/OS ISPF job error list panel EQQMEP1L

Job error handling can also be performed from the Job Scheduling Console. Figure 11-41 shows an example of of a job list with four jobs in error status. The job error codes are shown in column Error code.


WBJOBSTATUS1 job stream jobs - TWSC - Job Scheduling Console
Console
Selected
Edit
View
Window
Help
WBJOBSTATUS1 job stream jobs - TWSC
WBJOBSTATUS1 job stre...
WBJOBSTATUS1 jo
Ready to run list
WBJOBSTATUS1 job stream jobs - TWSC
IBM.

Job Stream Name	Job ID	Description	Task Name	Workstation	Status	Error Code	Job Number
WBJOBSTATUS1	10	Unix script rc=0	WBUNXRC0	TDWB	Successful		UNX97784
WBJOBSTATUS1	20	Unix script rc=8	WBUNXRC8	TDWB	Error	0008	UNX97174
WBJOBSTATUS1	30	JSDL job does not exist	WBNOJSDL	TDWB	Error	FAIL	
WBJOBSTATUS1	40	Wrong Unix user cred.	WBUNXUSR	TDWB	Error	FAIL	UNX95909
WBJOBSTATUS1	50	Unix script canceled	WBUNXCAN	TDWB	Error	0000	UNX93003

Total: 5 Displayed: 5 Selected: 0

Figure 11-41 Tivoli Workload Scheduler Job Scheduling Console job error list

Both Tivoli Workload Scheduler for z/OS error list panel EQQMEP1L and Job Scheduling Console job lists can be tailored by the user. Tailoring includes selecting columns and the column order to display.

Analysis and eventually recovery of Tivoli Workload Scheduler for z/OS end-to-end jobs in error status can be both simple and complex:

- A non-zero numerical error code indicates that the Tivoli Dynamic Workload Broker job execution took place, but was unsuccessful because a non-zero return code was returned to the operating system.
- An error code of 0000 or FAIL needs to be investigated further in order to identify error cause and recovery actions. Often the Tivoli Dynamic Workload Broker job status must be determined.

  Table 11-5 on page 610 shows how the different Tivoli Dynamic Workload Broker job status values map to Tivoli Workload Scheduler for z/OS job status codes and error codes.

11.7 Terminology

In this section the terminology used in this chapter is defined.

Note: In order to make the terminology used in this chapter we adopted a system of terminology that may be a bit different than that used in the product documentation.

11.7.1 Tivoli Workload Scheduler for z/OS end-to-end terminology

- IBM Tivoli Workload Scheduler suite

  The suite of programs that includes Tivoli Workload Scheduler and Tivoli Workload Scheduler. These programs are used together to make end-to-end scheduling work.

- IBM Tivoli Workload Scheduler

  The version of Tivoli Workload Scheduler that runs on UNIX, OS/400®, Linux, and Windows operating systems. Sometimes called IBM Tivoli Workload Scheduler Distributed.

- IBM Tivoli Workload Scheduler for z/OS

  The version of Tivoli Workload Scheduler that runs on z/OS (as distinguished from Tivoli Workload Scheduler by itself, without the for z/OS specification).

- Scheduling engine

  A Tivoli Workload Scheduler engine or Tivoli Workload Scheduler for z/OS engine.

- IBM Tivoli Workload Scheduler engine

  The part of Tivoli Workload Scheduler that does actual scheduling work, as distinguished from the other components that are related primarily to the user interface (for example, the Tivoli Workload Scheduler Connector).

- IBM Tivoli Workload Scheduler for z/OS engine

  The part of Tivoli Workload Scheduler for z/OS that does actual scheduling work, as distinguished from the other components that are related primarily to the user interface (for example, the Tivoli Workload Scheduler for z/OS Connector). Essentially, the controller plus the server.

- IBM Tivoli Workload Scheduler for z/OS controller

  This is the component that runs on the controlling z/OS system and that contains the tasks that manage the long term plan, the current plan, and the databases.

- IBM Tivoli Workload Scheduler for z/OS tracker

  The tracker acts as a communication link between the system it runs on and the controller.

- IBM Tivoli Workload Scheduler for z/OS server

  The part of Tivoli Workload Scheduler for z/OS that acts as master domain manager. It is based on the UNIX IBM Tivoli Workload Scheduler code and runs in z/OS UNIX System Services (USS) on the mainframe.

- Job Scheduling Console

  Job Scheduling Console (JSC), the common graphical user interface (GUI) to both IBM Tivoli Workload Scheduler and IBM Tivoli Workload Scheduler for z/OS scheduling engines.

- Tivoli Dynamic Workload Console

  The Tivoli Dynamic Workload Console is Web-based user interface for Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS. It provides you with a means of viewing and controlling scheduling activities in production on both the Tivoli Workload Scheduler distributed and the Tivoli Workload Scheduler for z/OS end-to-end environments.

- Master

  The top level of the Tivoli Workload Scheduler for z/OS end-to-end scheduling network. Also called the master domain manager because it is the domain manager of the MASTERDM (top-level) domain.

- Domain manager

  A fault-tolerant agent responsible for handling dependency resolution for subordinate agents. Essentially, a fault-tolerant agent with a few extra responsibilities.

- Backup domain manager

  A fault-tolerant agent or domain manager capable of assuming the responsibilities of its domain manager for automatic workload recovery.

- Fault-tolerant agent (FTA)

  An agent that keeps its own local copy of the symphony file. In the event of a loss of communication with the domain manager, the FTA continues operation and is capable of resolving local dependencies and launching its jobs without interruption. In Tivoli Workload Scheduler for z/OS end-to-end, FTAs are also referred to as *fault-tolerant workstations*.

- Standard agent

  Standard agents receive a light version of the symphony file containing only domain, workstation, and user definitions. Standard agents connected to a domain manager receive the request and information to run a job from the domain manager. Standard agents connected directly to Tivoli Workload Scheduler for z/OS server receive the request and information to run a job from the Tivoli Workload Scheduler for z/OS controller.

- Extended agent

  A logical entity hosted physically by fault-tolerant or standard agent that enables you to launch and control jobs on other systems and applications, such as PeopleSoft, Oracle Applications, SAP, and MVS™ JES2, and JES3.

- Long term plan (LTP)

  A high-level plan for system activity that covers a period of at least one day, and not more than four years.

- Current plan (CP)

  A detailed plan or schedule of system activity that covers at least one minute, and not more than 21 days. Typically a current plan covers 12 to 48 hours.

- Symphony file

  In an end-to-end scheduling environment, the Tivoli Workload Scheduler for z/OS engine, in its role as master domain manager of the distributed network, creates the symphony file from the Tivoli Workload Scheduler for z/OS current plan. The symphony file contains all of the job streams that are defined to run on fault-tolerant workstations, standard agents, and extended agents.

- Workstation (WS)

  A unit, place, or group that performs specific data processing functions. A logical place where work occurs in an operations department. Tivoli Workload Scheduler for z/OS requires that you define the following characteristic for each workstation: the type of work it does (computer, printer, or general), the quantity of work it can handle at any particular time, and the times it is active. The activity that occurs at each workstation is called an operation. Activities on computer type workstations are also called jobs.

- Fault-tolerant workstation

  A Tivoli Workload Scheduler for z/OS end-to-end workstation used to schedule jobs on a fault-tolerant agent in the distributed network.

11.7.2 Tivoli Dynamic Workload Broker terminology

The following terms are defined:

- Tivoli Dynamic Workload Broker

  The Tivoli Dynamic Workload Broker program product.

- Tivoli Dynamic Workload Broker server

  Performs the Tivoli Dynamic Workload Broker job management and resource management activities.

- Tivoli Workload Scheduler agent

  Transfers job requests received from a Tivoli Workload Scheduler for z/OS end-to-end domain manager to the Tivoli Dynamic Workload Broker server. It emulates the behavior of a Tivoli Workload Scheduler standard agent.

- Workload agent

  Receives job submission requests from the Tivoli Dynamic Workload Broker server job dispatcher and manages the running of Tivoli Dynamic Workload Broker jobs from start to finish on the local system.

- Tivoli Dynamic Workload Broker job

  A job definition containing all information and parameters necessary to run a Tivoli Dynamic Workload Broker job. It is a text file in .JSDL format.

- Job Submission Description Language (JSDL) file

  Synonym for a Tivoli Dynamic Workload Broker job.

- Job repository

  A Tivoli Dynamic Workload Broker server data store containing Tivoli Dynamic Workload Broker job definitions.

- Job Brokering Definition Console

  A structured editing tool you use to create and modify Job Submission Description Language (JSDL) files.

A

Using Tivoli Dynamic Workload Broker with Enterprise Workload Manager

This appendix describes how to configure Tivoli Dynamic Workload Broker and Enterprise Workload Manager to be used together for efficient job dispatching and scheduling. It reviews general configuration and interaction between the products, including Tivoli Dynamic Workload Broker job definition for Enterprise Workload Manager interaction, Enterprise Workload Manager classification, Enterprise Workload Manager load balancing, resource management, and monitoring of the work.

All information given in this book refers to Tivoli Dynamic Workload Broker Version 1.1 and Enterprise Workload Manager Version 1.2.

Note: This appendix is based on a whitepaper written by Alan Bivens, IBM USA, and Tullio Tancredi, IBM Italy.

IBM Enterprise Workload Manager

Enterprise Workload Manager is a robust performance management tool that allows the administrator to monitor and manage work that runs within enterprise environments. Enterprise Workload Manager has a variety of management abilities in its repertoire including the following:

- Advising load balancers (This management action is the focus of this book.)
- Moving CPU resources across virtual servers
- Provisioning new servers to needy server tiers

Enterprise Workload Manager management domains consist of a collection of Enterprise Workload Manager managed servers and a domain manager. The domain manager is the central point of control for a domain because it coordinates the activation of policies on the managed servers and the collection of performance data. A managed server is a server whose work requests are monitored by Enterprise Workload Manager. The managed server sends performance data to the domain manager. The domain manager includes the Enterprise Workload Manager user interface, referred to as the Enterprise Workload Manager Control Center. See Figure A-1.

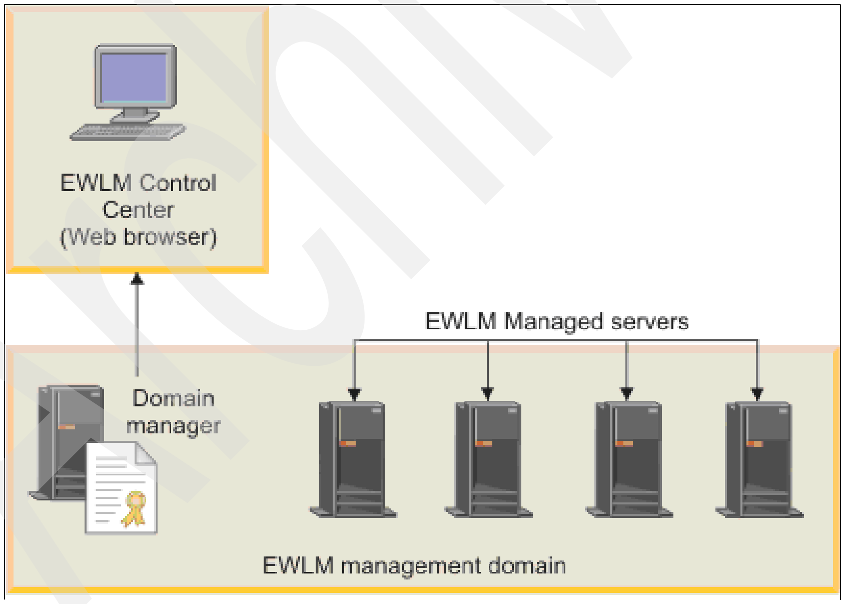


Figure A-1 Typical Enterprise Workload Manager environment

The Enterprise Workload Manager Managed Server gathers system statistics from the underlying OS. If the applications running on the system are Application Response Measurement (ARM) instrumented, the managed server will also gather application statistics. The Tivoli Dynamic Workload Broker agent is ARM instrumented to provide application information for Enterprise Workload Manager to use when calculating weights for efficient job dispatching.

Enterprise Workload Manager and Tivoli Dynamic Workload Broker interact in three different ways:

- Monitoring: The Tivoli Dynamic Workload Broker agent is ARM instrumented to allow Enterprise Workload Manager to classify and closely monitor the work running on the agent systems.
- Advanced load balancing/job dispatching: Tivoli Dynamic Workload Broker communicates with Enterprise Workload Manager through the Server/Application State Protocol (SASP) to get weights that indicate the best distribution of Tivoli Dynamic Workload Broker agents to which to route incoming jobs.
- Enterprise Workload Manager Autonomic Management: If on a supported autonomic management platform, Enterprise Workload Manager will use other resource allocation methods to help achieve the administratively configured goals of the Tivoli Dynamic Workload Broker jobs. See “Enterprise Workload Manager resource allocation for meeting job goals” on page 656.

If you are unfamiliar with Enterprise Workload Manager, we suggest the following sources for more information

- *IBM Enterprise Workload Manager*, SG24-6350
- Enterprise Workload Manager Infocenter:

  http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp

Planning for Tivoli Dynamic Workload Broker/Enterprise Workload Manager interaction

When planning for the interaction of Tivoli Dynamic Workload Broker and Enterprise Workload Manager, one must keep in mind the supported platforms of each product and the communication between them to ensure that the environment is supported by both products.

Platform support

The Tivoli Dynamic Workload Broker Server is not a job-processing entity. Thus, it is not ARM instrumented and does not need to be on a Enterprise Workload Manager monitored machine. The Agent Manager is also not a job-processing entity and need not be on an Enterprise Workload Manager monitored machine. The Tivoli Dynamic Workload Broker agent is ARM instrumented and processes the Tivoli Dynamic Workload Broker jobs. Therefore, it should be on a machine monitored with an Enterprise Workload Manager Managed Server. Because the Tivoli Dynamic Workload Broker agent will also be a part of the Enterprise Workload Manager Load Balancing algorithm, it should be on a machine with an OS with Enterprise Workload Manager Managed Server load balancing support. Given these requirements, it is important that the Tivoli Dynamic Workload Broker agent and Enterprise Workload Manager Managed Server be on the systems supported by the appropriate components of both products. A matrix of this cross-product support (Enterprise Workload Manager Managed Server load-balancing support and Tivoli Dynamic Workload Broker agent support) is in Table A-1.

Table A-1 Matrix of cross-product support

	IBM AIX 5L™ v5.2	**IBM AIX 5D v5.3**	**SLES 8**	**SLES 9**	**RHEL 3.0 and 4.0**	**Windows Server 2000 Standard or Enterprise Edition**	**Windows Server 2003 Standard or Enterprise Edition**	**Windows Server 2003 Standard AMD64 /EM64T**	**HP-UX 11iV1 and Solaris ™ 9**
TDWB Agent	X	X	X	X	X	X	X	X	
EWLM MS	X	X		X		X	X		X
TDWB and EWLM	**X**	**X**		**X**		**X**	**X**		

Table A-1 was created from the supported OS list of Tivoli Dynamic Workload Broker Version 1.1 and EWLM Release V1R2. Consult appropriate documentation to determine supported operating systems for future releases of either product.

Communication between products

The Tivoli Dynamic Workload Broker server connects to the Enterprise Workload Manager Domain Manager to receive SASP load balancing recommendations. This is a standard TCP connection (typically on port 3860) for binary communication. If the Tivoli Dynamic Workload Broker server is in a less secure zone than the Domain Manager, special provisions would need to be made to ensure that the Tivoli Dynamic Workload Broker connection can go through the firewall to reach the Domain Manager.

Enterprise Workload Manager load balancing recommendations in Tivoli Dynamic Workload Broker

In this document we provide a scenario on Tivoli Dynamic Workload Broker's use of Enterprise Workload Manager Load Balancing weights for efficient job dispatching. An example environment involving this interaction can be found in the Figure A-2.

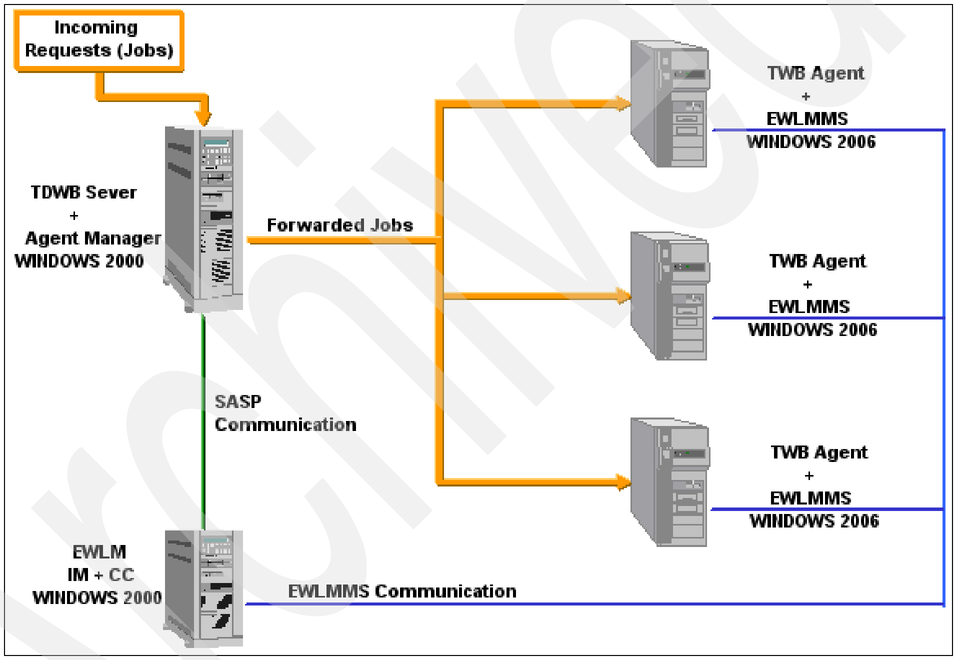


Figure A-2 Tivoli Dynamic Workload Broker using Enterprise Workload Manager Load Balancing weights

In Figure A-2, the flow of Tivoli Dynamic Workload Broker jobs is illustrated in the bold orange arrows. Enterprise Workload Manager's communication between its managed servers and its domain manager is shown in blue. The long-lived SASP connection between the Tivoli Dynamic Workload Broker Server and the Enterprise Workload Manager Domain Manager is shown in green. During the connection illustrated by the green line, the Tivoli Dynamic Workload Broker server registers the Tivoli Dynamic Workload Broker agents as group members with the Enterprise Workload Manager Domain Manager. The Tivoli Dynamic

Workload Broker Server then receives weight updates from the Domain Manager every 30 seconds. These weight updates are immediately applied and affect the manner in which Tivoli Dynamic Workload Broker dispatches jobs until the next weight update is received.

The following steps must be followed to start the interaction:

1. Enable Enterprise Workload Manager Load Balancing. See “Turning on Enterprise Workload Manager load balancing” on page 633.
2. Install the Enterprise Workload Manager Plug-in at Tivoli Dynamic Workload Broker install time. See “Enabling ARM on the Tivoli Dynamic Workload Broker agent” on page 636.
3. Set Enterprise Workload Manager as the optimization policy in the Tivoli Dynamic Workload Broker job definition. See “Job definitions” on page 636.

We highly recommend the remaining steps for the most efficient job dispatching interaction:

1. Enable ARM on the Tivoli Dynamic Workload Broker agent. See “Enabling ARM on the Tivoli Dynamic Workload Broker agent” on page 636.
2. Define Enterprise Workload Manager classification and service goal criteria. See “Enterprise Workload Manager classification of Tivoli Dynamic Workload Broker jobs” on page 637.

Starting the interaction

Now let us look at starting the interaction.

Note: Enterprise Workload Manager and Tivoli Dynamic Workload Broker integration troubleshooting is covered in 10.4, “Troubleshooting the integration with Enterprise Workload Manager” on page 512.

Turning on Enterprise Workload Manager load balancing

Enterprise Workload Manager's process of generating load balancing weights that may be used for the efficient dispatching of jobs must be turned on and configured prior to use. This can be done using the Configuration Wizard at Enterprise Workload Manager install time, or through the use of the **`changeDM`** command after Enterprise Workload Manager has already been installed. The typical configuration uses the following parameters:

-lbp (load balancing port)	3860
-ma (management address)*	0.0.0.0

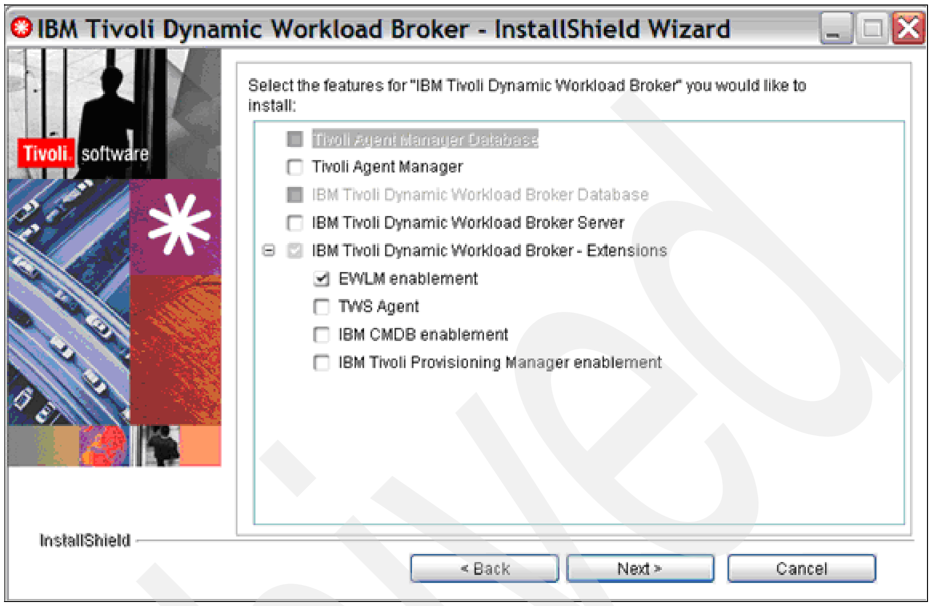
*A value of 0.0.0.0 is used for the management address to permit the domain manager to accept connections to the load balancing port on any valid IP address (this is important when using machines with more than one network interface).

Enabling Tivoli Dynamic Workload Broker to receive Enterprise Workload Manager Load Balancing weights

Next you have to enable Tivoli Dynamic Workload Broker to accept the load balancing weights from EWLM during the Tivoli Dynamic Workload Broker installation process.

Enterprise Workload Manager enablement is a Tivoli Dynamic Workload Broker extension that must be installed with the Tivoli Dynamic Workload Broker Server. When installed, the Enterprise Workload Manager plug-in for Tivoli Dynamic Workload Broker will be integrated into the server, registering the Tivoli Dynamic Workload Broker agents as load balancing group members in Enterprise Workload Manager and receiving new weights every 30 seconds.

As shown in Figure A-3, the administrator simply selects **EWLM enablement** from the feature list when installing Tivoli Dynamic Workload Broker.

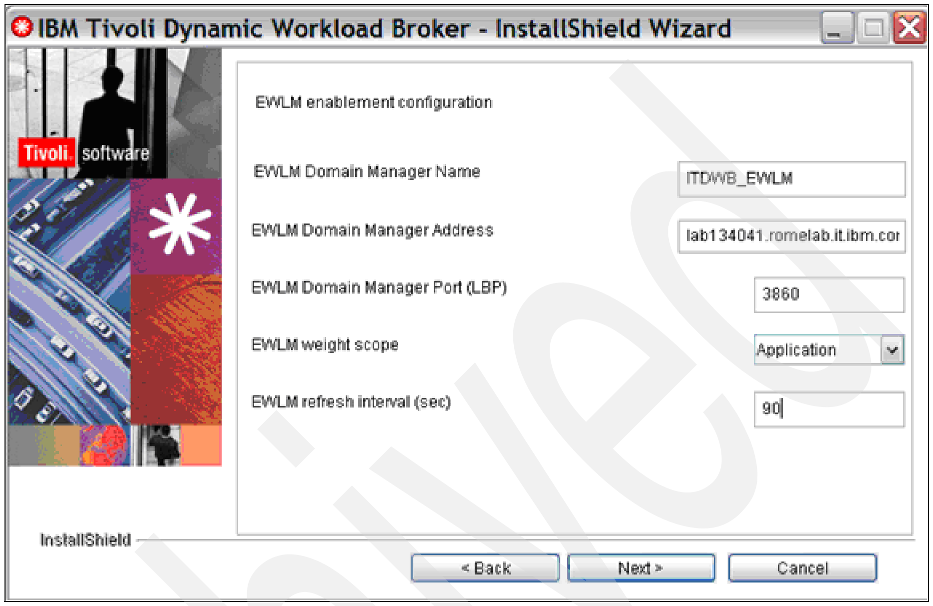


Select the features for "IBM Tivoli Dynamic Workload Broker" you would like to install:

- [■] Tivoli Agent Manager Database
- [ ] Tivoli Agent Manager
- [■] IBM Tivoli Dynamic Workload Broker Database
- [ ] IBM Tivoli Dynamic Workload Broker Server
- [✓] IBM Tivoli Dynamic Workload Broker - Extensions
  - [✓] EWLM enablement
  - [ ] TWS Agent
  - [ ] IBM CMDB enablement
  - [ ] IBM Tivoli Provisioning Manager enablement

Figure A-3 Tivoli Dynamic Workload Broker EWLM enablement

Once EWLM enablement is selected, the EWLM enablement configuration panel appears, allowing the administrator to tell Tivoli Dynamic Workload Broker how to connect to the EWLM Domain Manager. See Figure A-4.



EWLM enablement configuration

EWLM Domain Manager Name ITDWB_EWLM

EWLM Domain Manager Address lab134041.romelab.it.ibm.cor

EWLM Domain Manager Port (LBP) 3860

EWLM weight scope Application

EWLM refresh interval (sec) 90

Figure A-4 Connections to Enterprise Workload Manager

The following fields must be completed:

- EWLM Domain Manager Name: This is the unique identifier that the Tivoli Dynamic Workload Broker server will use to identify itself with the Enterprise Workload Manager Domain Manager. Care should be taken to ensure that this name will not be used by any other products connecting to EWLM for load-balancing weights.
- EWLM Domain Manager Address: This is the IP address or resolvable host name of the Enterprise Workload Manage Domain Manager. The value used here should be the same value used for the -ma parameter of the Enterprise Workload Manage installation (unless 0.0.0.0 was used for the -ma parameter).
- EWLM Domain Manager Port (LBP): This is the SASP port for the EWLM Domain Manager. The value used here should be the same value used in the -lbp parameter during Enterprise Workload Manager configuration.
- EWLM weight scope: This value should always be set to *Application*.
- EWLM refresh interval: This is the interval that determines how often the Tivoli Dynamic Workload Broker Server will try to update Domain Manager group settings (if needed). The default value is 90 seconds.

Job definitions

In order to make Tivoli Dynamic Workload Broker dispatch jobs using the weights provided by the Enterprise Workload Manager, it is necessary to define EWLM as the optimization policy in the job definition ,as shown in Example A-1.

Example A-1 Defining EWLM as the optimization policy

```
<jsdl:optimization name="JPT_EWLM">
   <jsdl:ewlm/>
</jsdl:optimization>
```

Enabling ARM on the Tivoli Dynamic Workload Broker agent

Enabling Application Response Measurement on the Tivoli Dynamic Workload Broker agent requires the following two steps:

1. Set the arm.enabled variable on the Job Execution Agent.

   Set arm.enabled=true in the JobExecutionAgent.properties file.

   The JobExecutionAgent.properties file can be found at the following path:

   `<`*`ITDWB_Agent_InstallDir`*`>\ep\runtime\agent\subagents`
   `\JobExecutionAgent\`

2. Set the ARM native library path correctly.

   The Tivoli Dynamic Workload Broker agent exploits IBM Java ARM implementation, which needs a native ARM library. Table A-2 describes the native library name and location for the collectively supported environments shown in "Platform support" on page 629.

Table A-2 Native library names and locations

Operating system	Library name	Library path
IBM AIX 32 bit	libewljarm4.a	/usr/lib/libewljarm4.a
IBM AIX 64 bit	libewljarm4_64.a	/usr/lib/libewljarm4_64.a
Microsoft Windows	ewljarm4.dll	%VE%\EWLM\class\ms\ewljarm4.so
SLES 9	libewljarm4.so	/usr/lib/libewljarm4.a

In order for Tivoli Dynamic Workload Broker to find this library, the native library path must be edited to include the directory of the appropriate library listed above.

- For AIX: The LD_LIBRARY_PATH environment is used to define the native library path (that is, LD_LIBRARY_PATH=/usr/lib).
- For Windows: The directory containing the JNI™ library should be defined by the PATH environment variable (that is, PATH=%PATH%;<ewlm_root>\ms).
- For SLES 9: The directory containing the JNI library should be defined by the PATH environment variable (that is, PATH=/usr/lib).

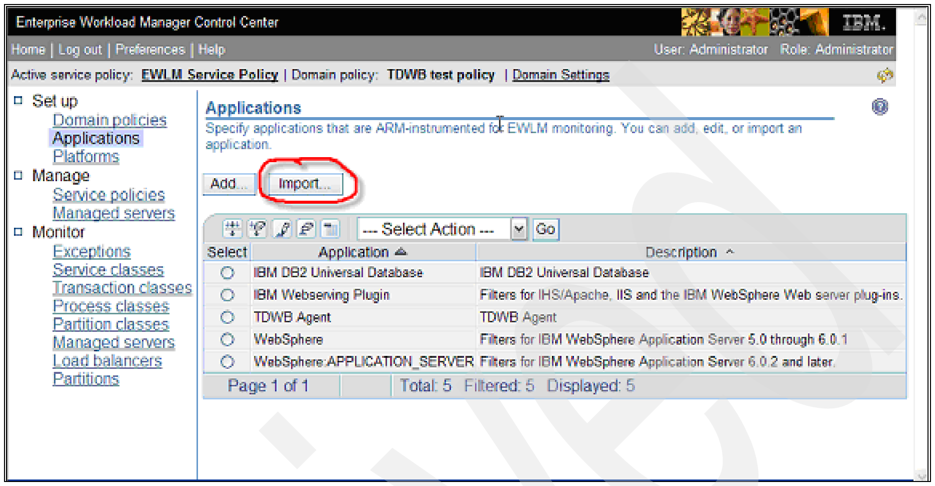
  If the IBM JNI native library could not be found, the Tivoli Dynamic Workload Broker agent will be not ARM instrumented. More information can be found in the appropriate ARM-related documentation from the Enterprise Workload Manager InfoCenter at:

  `http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp`

Enterprise Workload Manager classification of Tivoli Dynamic Workload Broker jobs

Enterprise Workload Manager will report unclassified Tivoli Dynamic Workload Broker work without any configuration from the administrator. However, Enterprise Workload Manager classifies work in the enterprise environment to understand management importance, identify work for goal setting, and to provide more detailed reporting. In order to classify work being done in different applications, Enterprise Workload Manager must first understand the information each instrumented application will expose for classification. This is provided in an Application filter xml file (named TDWB Agent.xml) currently available from Enterprise Workload Manager or Tivoli Dynamic Workload Broker product teams.

This file must be imported into the Enterprise Workload Manager Control Center by using the Import button on the Applications panel (the Applications panel is reached by using the Applications link under Set up). See Figure A-5.


Enterprise Workload Manager Control Center

Home | Log out | Preferences | Help   User: Administrator   Role: Administrator

Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings

Applications

Specify applications that are ARM-instrumented for EWLM monitoring. You can add, edit, or import an application.

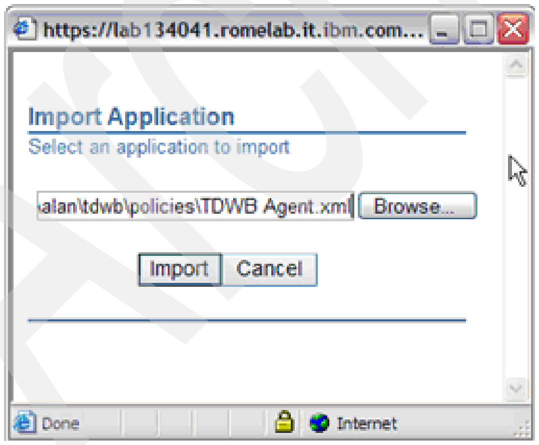
Select	Application	Description
○	IBM DB2 Universal Database	IBM DB2 Universal Database
○	IBM Webserving Plugin	Filters for IHS/Apache, IIS and the IBM WebSphere Web server plug-ins.
○	TDWB Agent	TDWB Agent
○	WebSphere	Filters for IBM WebSphere Application Server 5.0 through 6.0.1
○	WebSphere:APPLICATION_SERVER	Filters for IBM WebSphere Application Server 6.0.2 and later.

Page 1 of 1   Total: 5 Filtered: 5 Displayed: 5

Figure A-5 Importing TDWB_Agent.xml

Import the file named TDWB Agent.xml, as shown in Figure A-6. Tivoli Dynamic Workload Broker agent can be found at the following path in the Tivoli Dynamic Workload Broker Server directory space:

<*server_installation_directory*>/EWLM/samples


https://lab134041.romelab.it.ibm.com...

Import Application

Select an application to import

alan\tdwb\policies\TDWB Agent.xml

Figure A-6 TDWB_Agent.xml path

After the Tivoli Dynamic Workload Broker application file has been imported, you will then see it in the list of applications. This step will also allow the Tivoli

Dynamic Workload Broker agent to be used for classifying work in the Enterprise Workload Manager policy.

Create the Enterprise Workload Manager policy

There is extensive documentation on the general creation of Enterprise Workload Manager policies. See the Enterprise Workload Manager documentation at:

http://publib.boulder.ibm.com/infocenter/eserver/v1r1/en_US/index.htm?info/ewlminfo/kickoff.htm

Therefore, we only cover the parts pertaining to the Tivoli Dynamic Workload Broker interaction. Before defining Tivoli Dynamic Workload Broker classification criteria in the Enterprise Workload Manager policy for work in the current environment, the application should also be added to the policy as one of the applications that can classify work for this policy. This is done by using the **Add** button in the Applications panel of the policy creation screens. (The panel can be reached by selecting the **Domain Policies** link under Set up, and then creating a new or editing an existing Domain Policy). See Figure A-7.

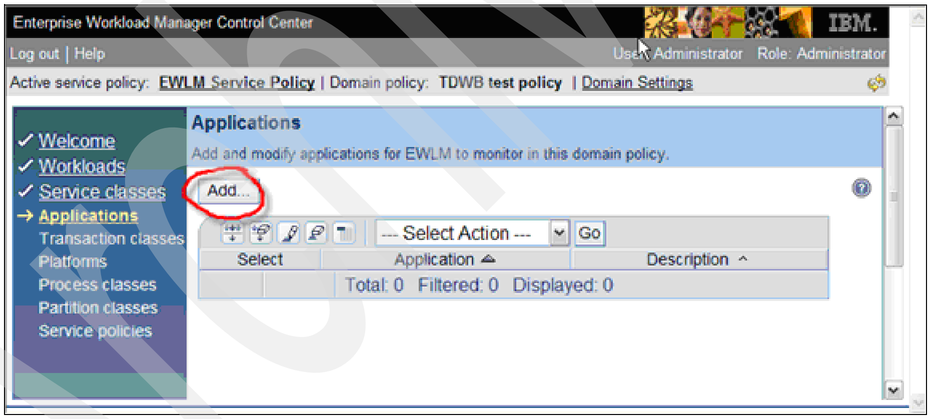
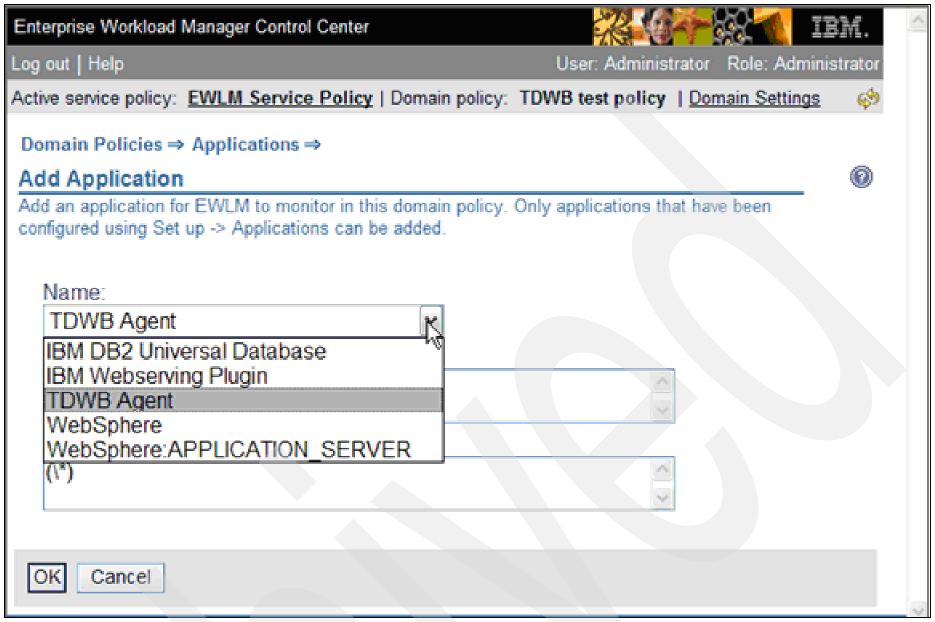

Enterprise Workload Manager Control Center
IBM.
Log out | Help
User: Administrator Role: Administrator
Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings
Welcome
Workloads
Service classes
Applications
Transaction classes
Platforms
Process classes
Partition classes
Service policies
Applications
Add and modify applications for EWLM to monitor in this domain policy.
Add...
--- Select Action ---
Go
Select
Application
Description
Total: 0 Filtered: 0 Displayed: 0

Figure A-7 Adding an application that can classify work

Select the Tivoli Dynamic Workload Broker agent application to add from the drop-down menu, and then click **OK,** as shown in Figure A-8.


Enterprise Workload Manager Control Center

Log out | Help   User: Administrator   Role: Administrator

Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings

Domain Policies ⇒ Applications ⇒

Add Application

Add an application for EWLM to monitor in this domain policy. Only applications that have been configured using Set up -> Applications can be added.

Name: TDWB Agent

- IBM DB2 Universal Database
- IBM Webserving Plugin
- TDWB Agent
- WebSphere
- WebSphere:APPLICATION_SERVER

(*)

Figure A-8 Drop-down menu shows application to add

Now that the Tivoli Dynamic Workload Broker agent application has been added to the policy, you will be able to select it as one of the applications used to classify work when creating Enterprise Workload Manager service classes and transaction classes.

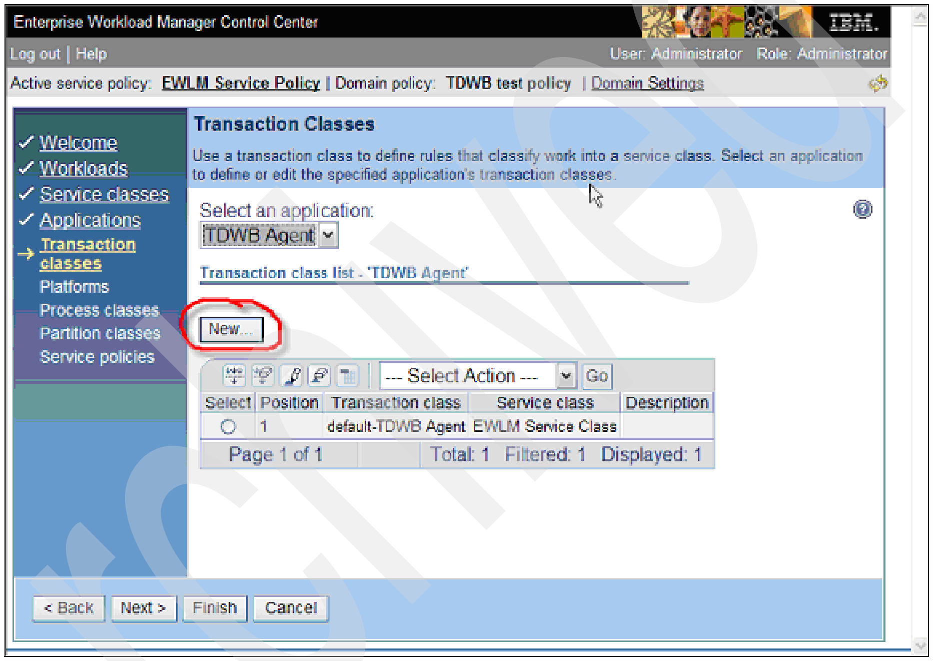
Create Enterprise Workload Manager service classes

Much of the remainder of this document focuses on how to classify the Tivoli Dynamic Workload Broker work into Enterprise Workload Manager transaction classes. However, each transaction class must be in a Enterprise Workload Manager service class in order to be actively managed by Enterprise Workload Manager. The service classes must exist prior to the creation of the transaction classes. The service classes for Tivoli Dynamic Workload Broker work are no different from any other service classes defined in Enterprise Workload Manager about the creation of Enterprise Workload Manager service classes.

While Enterprise Workload Manager permits a variety of goal types to be used in its service classes, some of the most popular goal types for transaction classes are response time oriented goals. These goals can be very effective for short transactions or for those that must be executed in a particular time frame. However, administrators may find that velocity goals specifying the comparative fraction of resources for the service class may be more appropriate for longer running transactions.

Create Enterprise Workload Manager transaction classes

Once the TDWB Agent application has been added and service classes have been created, transaction classes can now be created. The Transaction Class creation panel is shown in Figure A-9. Note that the TDWB application is selected from the drop-down menu and Enterprise Workload Manager has already created a default transaction class. The New button should be used to create new transaction classes with TDWB classification criteria.



Transaction Classes

Use a transaction class to define rules that classify work into a service class. Select an application to define or edit the specified application's transaction classes.

Select an application: TDWB Agent

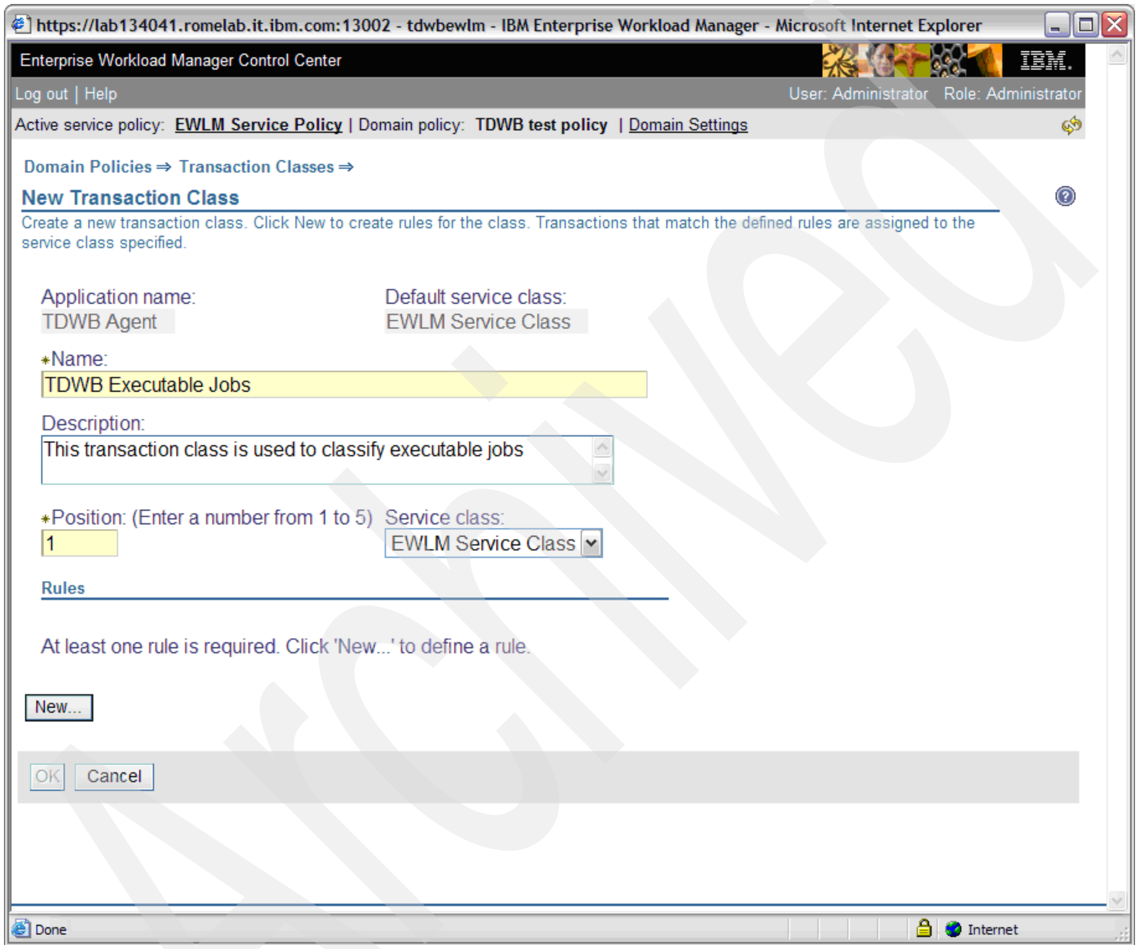
Transaction class list - 'TDWB Agent'

Select	Position	Transaction class	Service class	Description
○	1	default-TDWB Agent	EWLM Service Class	

Page 1 of 1 Total: 1 Filtered: 1 Displayed: 1

Figure A-9 Transaction class creation panel

After selecting the **New** button, the New Transaction Class panel will be displayed, giving the administrator the ability to specify the transaction class name, the position of classification, and the associated service class. To begin setting the classification rules, the **New** button under Rules must be selected. See Figure A-10.



https://lab134041.romelab.it.ibm.com:13002 - tdwbewlm - IBM Enterprise Workload Manager - Microsoft Internet Explorer

Enterprise Workload Manager Control Center

Log out | Help   User: Administrator Role: Administrator

Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings

Domain Policies ⇒ Transaction Classes ⇒

New Transaction Class

Create a new transaction class. Click New to create rules for the class. Transactions that match the defined rules are assigned to the service class specified.

Application name: TDWB Agent

Default service class: EWLM Service Class

*Name: TDWB Executable Jobs

Description: This transaction class is used to classify executable jobs

*Position: (Enter a number from 1 to 5) 1

Service class: EWLM Service Class

Rules

At least one rule is required. Click 'New...' to define a rule.

Figure A-10 Start setting classification rules

Creating rules for the Tivoli Dynamic Workload Broker/Enterprise Workload Manager interaction is covered in detail in “Create Enterprise Workload Manager transactions classified by Tivoli Dynamic Workload Broker application name” on page 644; “Create Enterprise Workload Manager transactions classified by job name” on page 646; and “Create Enterprise Workload Manager transactions by categories” on page 650.

Tivoli Dynamic Workload Broker/Enterprise Workload Manager joint classification criteria

Classification of the Tivoli Dynamic Workload Broker work can be done in several ways. Table A-3 briefly describes the current criteria that may be used for classification of Tivoli Dynamic Workload Broker work (each will be explained in detail in later subsections).

Table A-3 Criteria for classification of Tivoli Dynamic Workload Broker work

	EWLM classification criteria name	**TDWB job definition criteria name**	**Valid TDWB values**
Work type	EWLM:Transaction name	<jsdl:application name = "…">	► "executable ► "j2ee ► "<any new pluggable application type>
Job name	JobName	<jsdl:jobDefinition name = "…">	<user entered string value>
Categories *order dependent	► Category1 ► Category2 ► Category3	► <jsdl:category>…</jsdl:category> ► "<jsdl:category>…</jsdl:category> ► "<jsdl:category>…</jsdl:category>	<user entered string values>

Note that the Category classification criteria is named Category1, Category2, and Category3 in Enterprise Workload Manager, but has no number when first described in the Tivoli Dynamic Workload Broker job definition. When using categories in the Tivoli Dynamic Workload Broker job definition, the first Tivoli Dynamic Workload Broker category defined is Enterprise Workload Manager's Category1, the second Tivoli Dynamic Workload Broker Category is Enterprise Workload Manager's Category2, and so on. This translation between products is done automatically.

It is not necessary to make Enterprise Workload Manager transaction classes using every way of classifying Tivoli Dynamic Workload Broker work. The three different classification methods are present to provide maximum flexibility to the administrator.

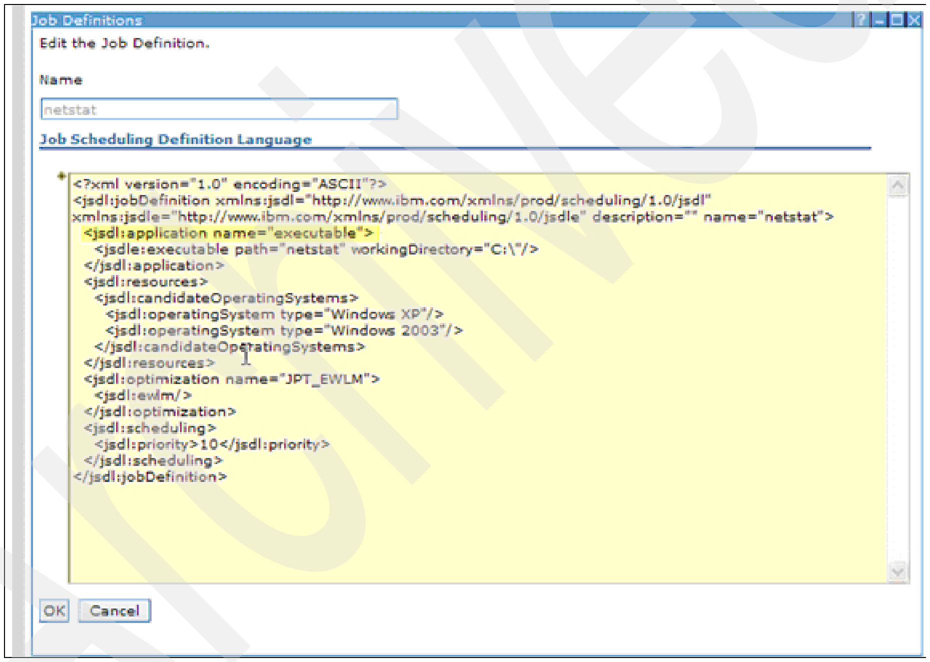
- One could create a silver transaction class, which is a combination of any of these three, for example, all work executable work (EWLM:Transaction name = executable) with job name of "silver" (JobName = "silver").
- One could create a silver transaction class that is simply based on one of the criteria, for example, all work with job name of "silver" (JobName = "silver").

Create Enterprise Workload Manager transactions classified by Tivoli Dynamic Workload Broker application name

Tivoli Dynamic Workload Broker allows its users to create different types of jobs to be executed by the Tivoli Dynamic Workload Broker agents. The two types currently supported are:

- Executable
- j2ee

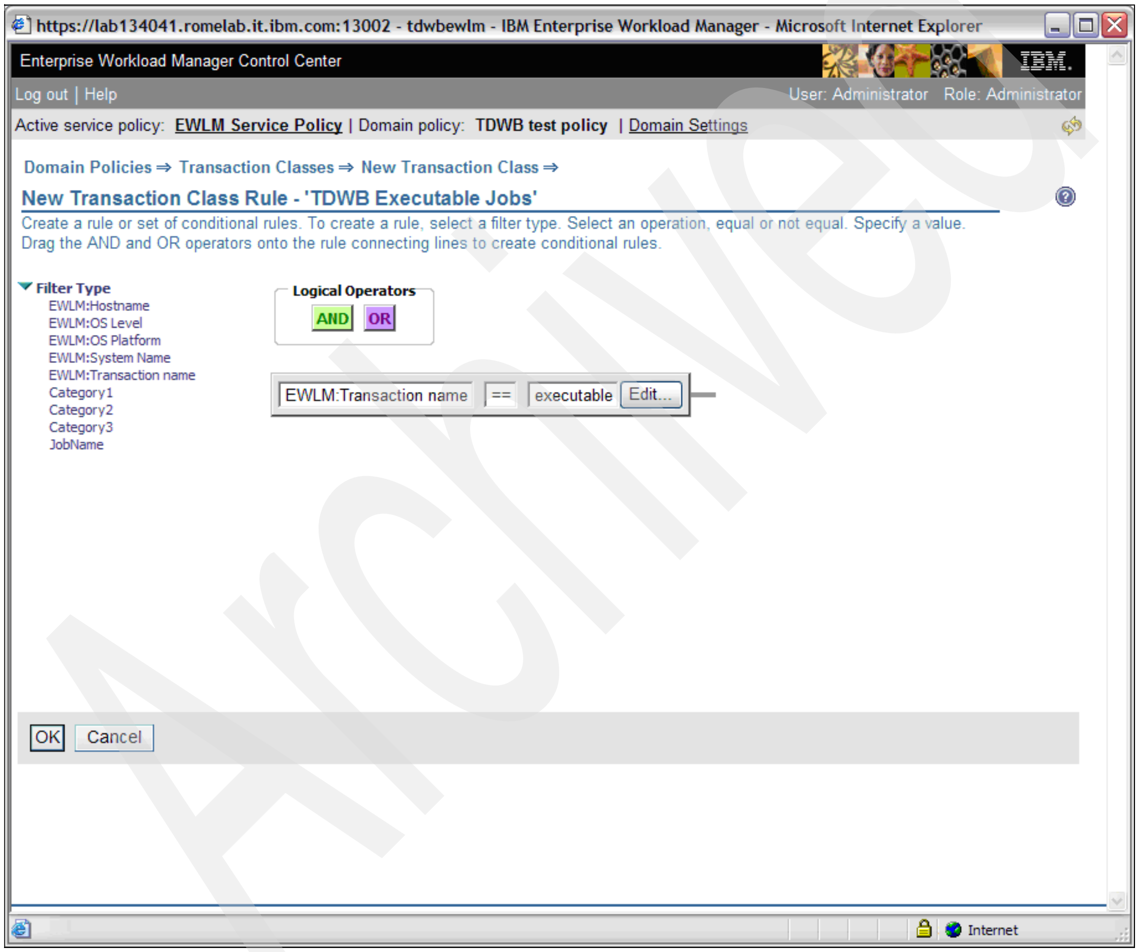
If the user would like to add new job types, he may create a pluggable bundle for Tivoli Dynamic Workload Broker and specify this job type in the Job Definition. The type of job must be provided as a name attribute of the application element in the Job Definition file. See Figure A-11.



```
<?xml version="1.0" encoding="ASCII"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" description="" name="netstat">
  <jsdl:application name="executable">
    <jsdle:executable path="netstat" workingDirectory="C:\"/>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="Windows XP"/>
      <jsdl:operatingSystem type="Windows 2003"/>
    </jsdl:candidateOperatingSystems>
  </jsdl:resources>
  <jsdl:optimization name="JPT_EWLM">
    <jsdl:ewlm/>
  </jsdl:optimization>
  <jsdl:scheduling>
    <jsdl:priority>10</jsdl:priority>
  </jsdl:scheduling>
</jsdl:jobDefinition>
```

Figure A-11   Tivoli Dynamic Workload Broker application name

Enterprise Workload Manager recognizes this Tivoli Dynamic Workload Broker application name as a transaction type. To create an Enterprise Workload Manager rule using the Tivoli Dynamic Workload Broker application name, the administrator needs to select **EWLM:Transaction name** to be in the left side of the rule, the "=" for the operator, and enter the appropriate Tivoli Dynamic Workload Broker application name value for the right side of the rule equation. Figure A-12 shows a rule defined using an EWLM Transaction name equal to *executable*. (See other possible Tivoli Dynamic Workload Broker application name values in Table A-3 on page 643.)


https://lab134041.romelab.it.ibm.com:13002 - tdwbewlm - IBM Enterprise Workload Manager - Microsoft Internet Explorer

Enterprise Workload Manager Control Center

Log out | Help

User: Administrator Role: Administrator

Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings

Domain Policies ⇒ Transaction Classes ⇒ New Transaction Class ⇒

New Transaction Class Rule - 'TDWB Executable Jobs'

Create a rule or set of conditional rules. To create a rule, select a filter type. Select an operation, equal or not equal. Specify a value. Drag the AND and OR operators onto the rule connecting lines to create conditional rules.

Filter Type
- EWLM:Hostname
- EWLM:OS Level
- EWLM:OS Platform
- EWLM:System Name
- EWLM:Transaction name
- Category1
- Category2
- Category3
- JobName

Logical Operators: AND OR

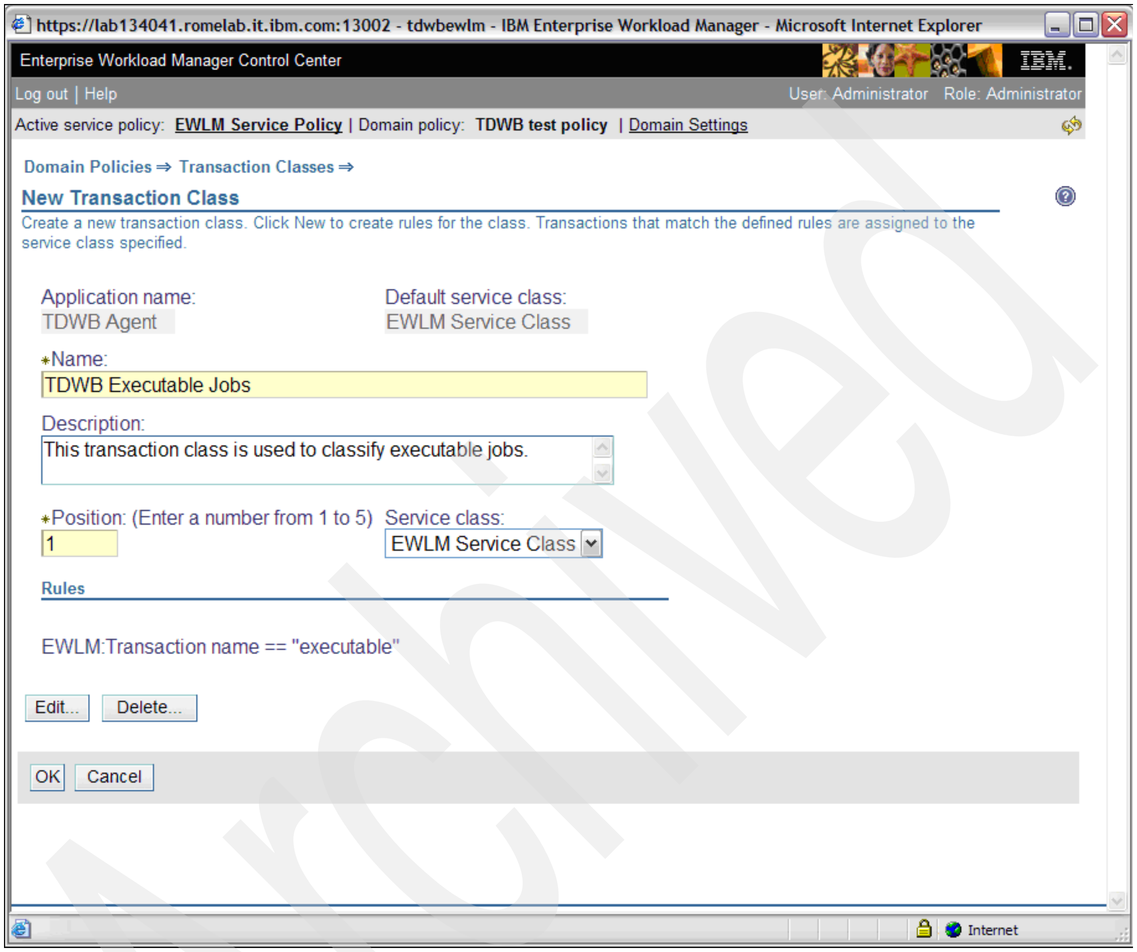
EWLM:Transaction name == executable Edit...

OK Cancel

Internet

Figure A-12 Creating a rule using EWLM transaction name and Tivoli Dynamic Workload Broker application name

The new rule will be displayed in the final panel for the Transaction Class definition, as shown in Figure A-13.


https://lab134041.romelab.it.ibm.com:13002 - tdwbewlm - IBM Enterprise Workload Manager - Microsoft Internet Explorer

Enterprise Workload Manager Control Center

Log out | Help   User: Administrator   Role: Administrator

Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings

Domain Policies ⇒ Transaction Classes ⇒

New Transaction Class

Create a new transaction class. Click New to create rules for the class. Transactions that match the defined rules are assigned to the service class specified.

Application name: TDWB Agent

Default service class: EWLM Service Class

*Name: TDWB Executable Jobs

Description: This transaction class is used to classify executable jobs.

*Position: (Enter a number from 1 to 5) 1

Service class: EWLM Service Class

Rules

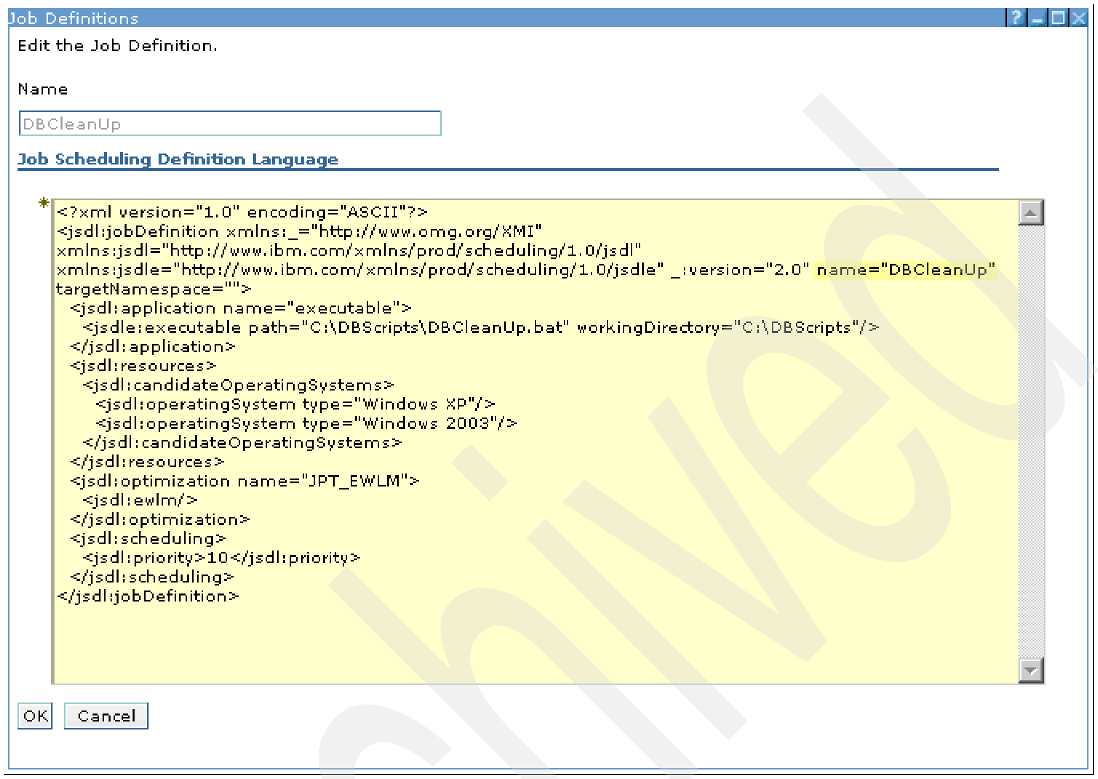
EWLM:Transaction name == "executable"

Figure A-13 Displaying new transaction name rule in transaction class panel

Create Enterprise Workload Manager transactions classified by job name

Classifying using a Tivoli Dynamic Workload Broker job name is similar to creating a transaction class using the Tivoli Dynamic Workload Broker application name attribute (described in “Create Enterprise Workload Manager transactions classified by Tivoli Dynamic Workload Broker application name” on page 644). First, the job name should be configured in the Tivoli Dynamic

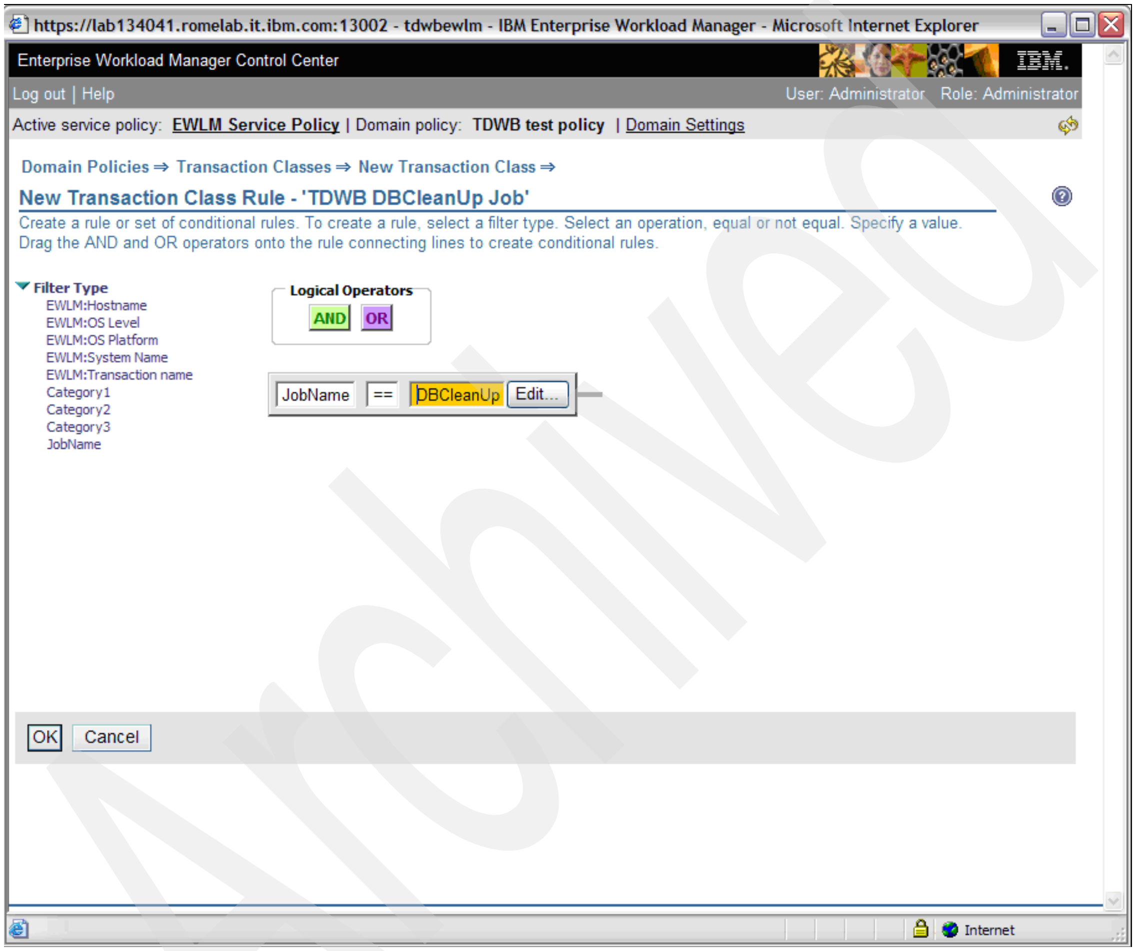
Workload Broker Job Definition. In Figure A-15, the job name of DBCleanUp has been configured.



```
<?xml version="1.0" encoding="ASCII"?>
<jsdl:jobDefinition xmlns:_="http://www.omg.org/XMI"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" _:version="2.0" name="DBCleanUp"
targetNamespace="">
  <jsdl:application name="executable">
    <jsdle:executable path="C:\DBScripts\DBCleanUp.bat" workingDirectory="C:\DBScripts"/>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="Windows XP"/>
      <jsdl:operatingSystem type="Windows 2003"/>
    </jsdl:candidateOperatingSystems>
  </jsdl:resources>
  <jsdl:optimization name="JPT_EWLM">
    <jsdl:ewlm/>
  </jsdl:optimization>
  <jsdl:scheduling>
    <jsdl:priority>10</jsdl:priority>
  </jsdl:scheduling>
</jsdl:jobDefinition>
```

Figure A-14 Tivoli Dynamic Workload Broker job name

To create an Enterprise Workload Manager rule using the Tivoli Dynamic Workload Broker job name, the administrator needs to select **Job Name** to be in the left side of the rule, the "=" for the operator, and enter the configured Tivoli Dynamic Workload Broker Job Name for the right side of the rule equation. (In the example below, the job name value is DBCleanUp.) See Figure A-15.



https://lab134041.romelab.it.ibm.com:13002 - tdwbewlm - IBM Enterprise Workload Manager - Microsoft Internet Explorer

Enterprise Workload Manager Control Center

Log out | Help   User: Administrator   Role: Administrator

Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings

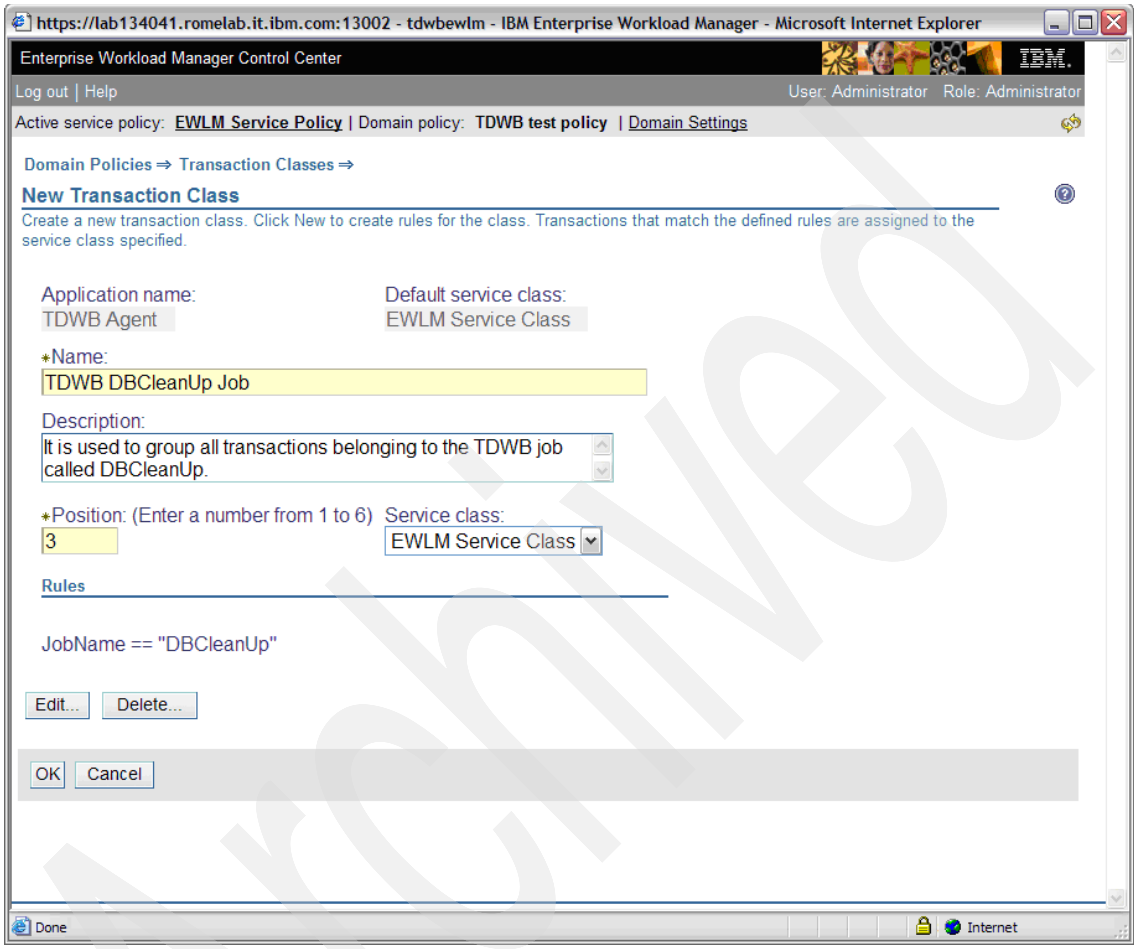
Domain Policies ⇒ Transaction Classes ⇒ New Transaction Class ⇒

New Transaction Class Rule - 'TDWB DBCleanUp Job'

Create a rule or set of conditional rules. To create a rule, select a filter type. Select an operation, equal or not equal. Specify a value. Drag the AND and OR operators onto the rule connecting lines to create conditional rules.

Figure A-15 Creating a rule using job name

The resulting transaction class and newly created rule are then shown in Figure A-16.


https://lab134041.romelab.it.ibm.com:13002 - tdwbewlm - IBM Enterprise Workload Manager - Microsoft Internet Explorer

Enterprise Workload Manager Control Center

Log out | Help   User: Administrator Role: Administrator

Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings

Domain Policies ⇒ Transaction Classes ⇒

New Transaction Class

Create a new transaction class. Click New to create rules for the class. Transactions that match the defined rules are assigned to the service class specified.

Application name: TDWB Agent	Default service class: EWLM Service Class

*Name: TDWB DBCleanUp Job

Description: It is used to group all transactions belonging to the TDWB job called DBCleanUp.

*Position: (Enter a number from 1 to 6): 3   Service class: EWLM Service Class

Rules

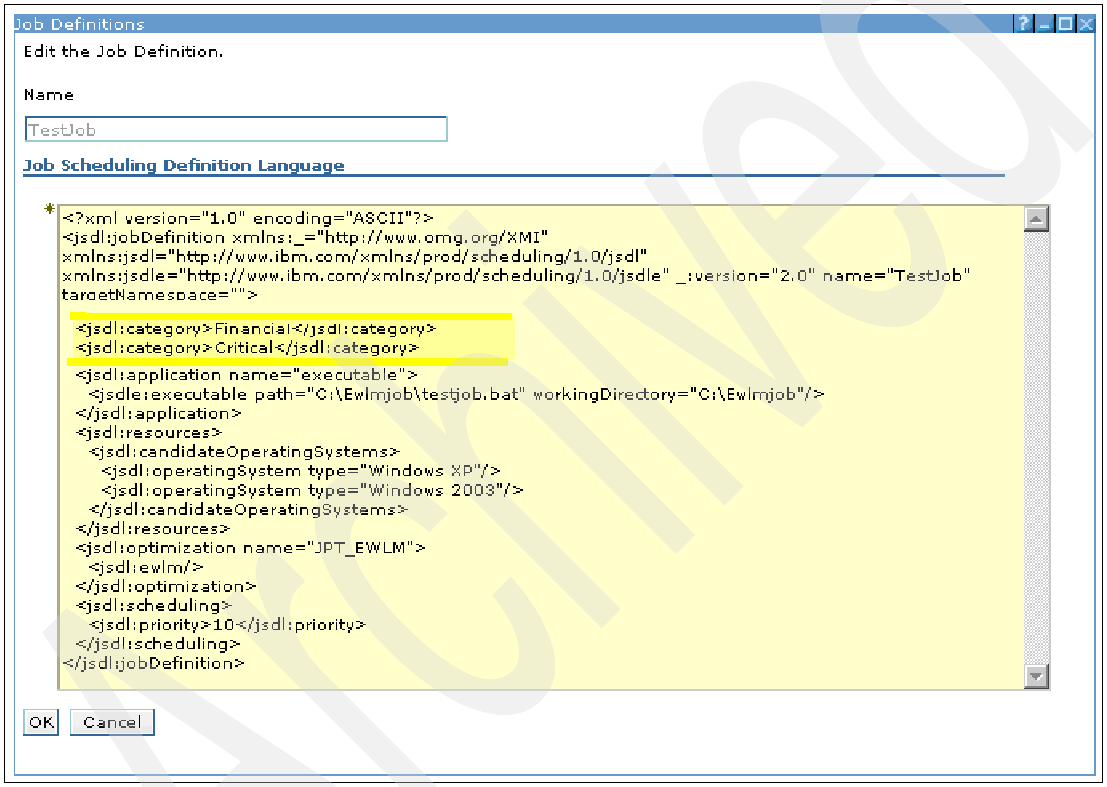
JobName == "DBCleanUp"

Figure A-16 Displaying new job name rule in transaction class panel

Create Enterprise Workload Manager transactions by categories

In addition to Tivoli Dynamic Workload Broker application name and job name, Tivoli Dynamic Workload Broker permits administrators to classify jobs using the category attribute in the job definition. It is possible to define up to three different categories for each job. Figure A-17 shows the definition of a job belonging to two different categories:

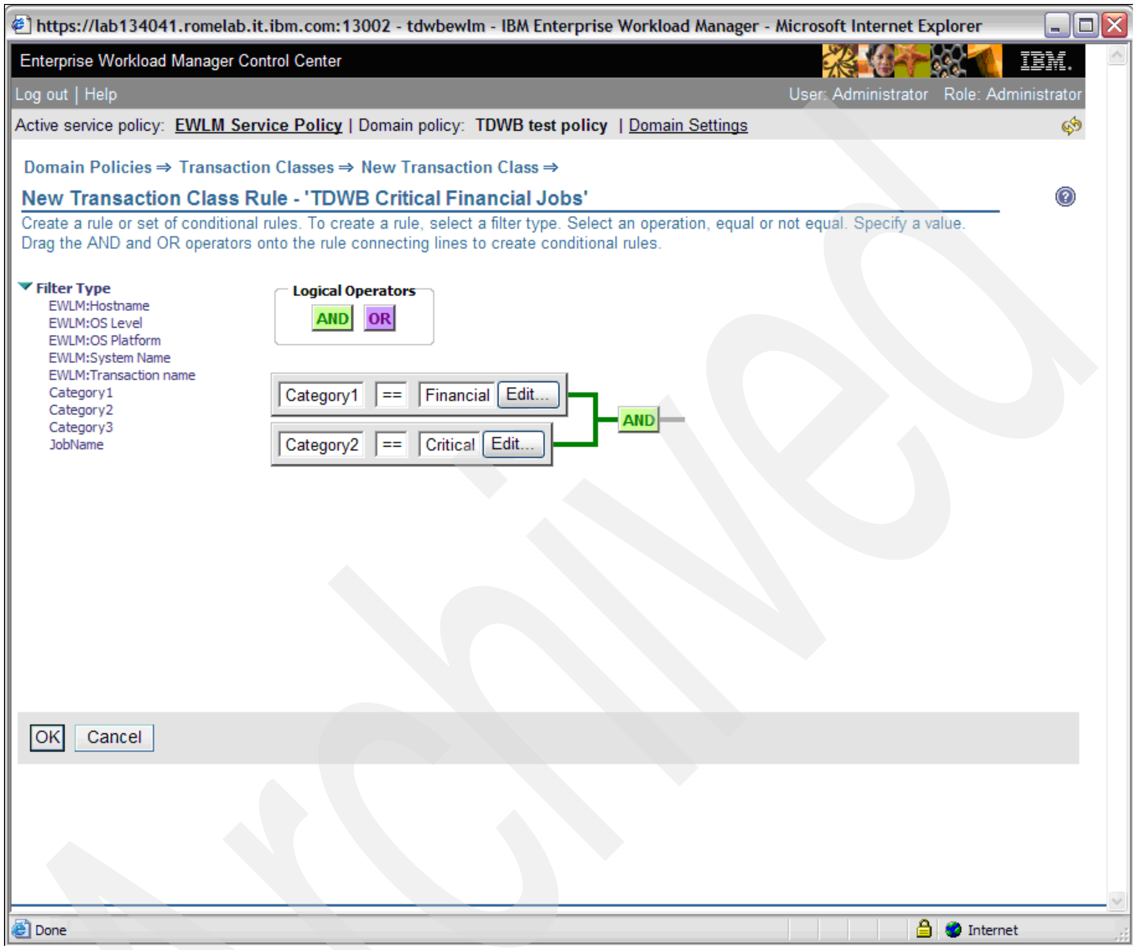
- It is a *Financial* job.
- It is a *Critical* job.



```
<?xml version="1.0" encoding="ASCII"?>
<jsdl:jobDefinition xmlns:_="http://www.omg.org/XMI"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" _:version="2.0" name="TestJob"
targetNamespace="">
  <jsdl:category>Financial</jsdl:category>
  <jsdl:category>Critical</jsdl:category>
  <jsdl:application name="executable">
    <jsdle:executable path="C:\Ewlmjob\testjob.bat" workingDirectory="C:\Ewlmjob"/>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="Windows XP"/>
      <jsdl:operatingSystem type="Windows 2003"/>
    </jsdl:candidateOperatingSystems>
  </jsdl:resources>
  <jsdl:optimization name="JPT_EWLM">
    <jsdl:ewlm/>
  </jsdl:optimization>
  <jsdl:scheduling>
    <jsdl:priority>10</jsdl:priority>
  </jsdl:scheduling>
</jsdl:jobDefinition>
```

Figure A-17 Tivoli Dynamic Workload Broker categories

To create an Enterprise Workload Manager rule using Tivoli Dynamic Workload Broker categories, the administrator needs to select the appropriate Enterprise Workload Manager category for the left side of the rule equation. If multiple Tivoli Dynamic Workload Broker categories are used, rules combining the appropriate Enterprise Workload Manager categories can be created. In the example below, Category1 was first selected with a value of Financial on the right side of the rule equation. The logical AND operator is then dragged over to the end of the rule equation to create the opportunity for a second, joint rule. In the second rule, Category2 was selected to be in the left side of the rule and the value of Critical is used in the right side of the rule. See Figure A-18.


https://lab134041.romelab.it.ibm.com:13002 - tdwbewlm - IBM Enterprise Workload Manager - Microsoft Internet Explorer

Enterprise Workload Manager Control Center

Log out | Help   User: Administrator   Role: Administrator

Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings

Domain Policies ⇒ Transaction Classes ⇒ New Transaction Class ⇒

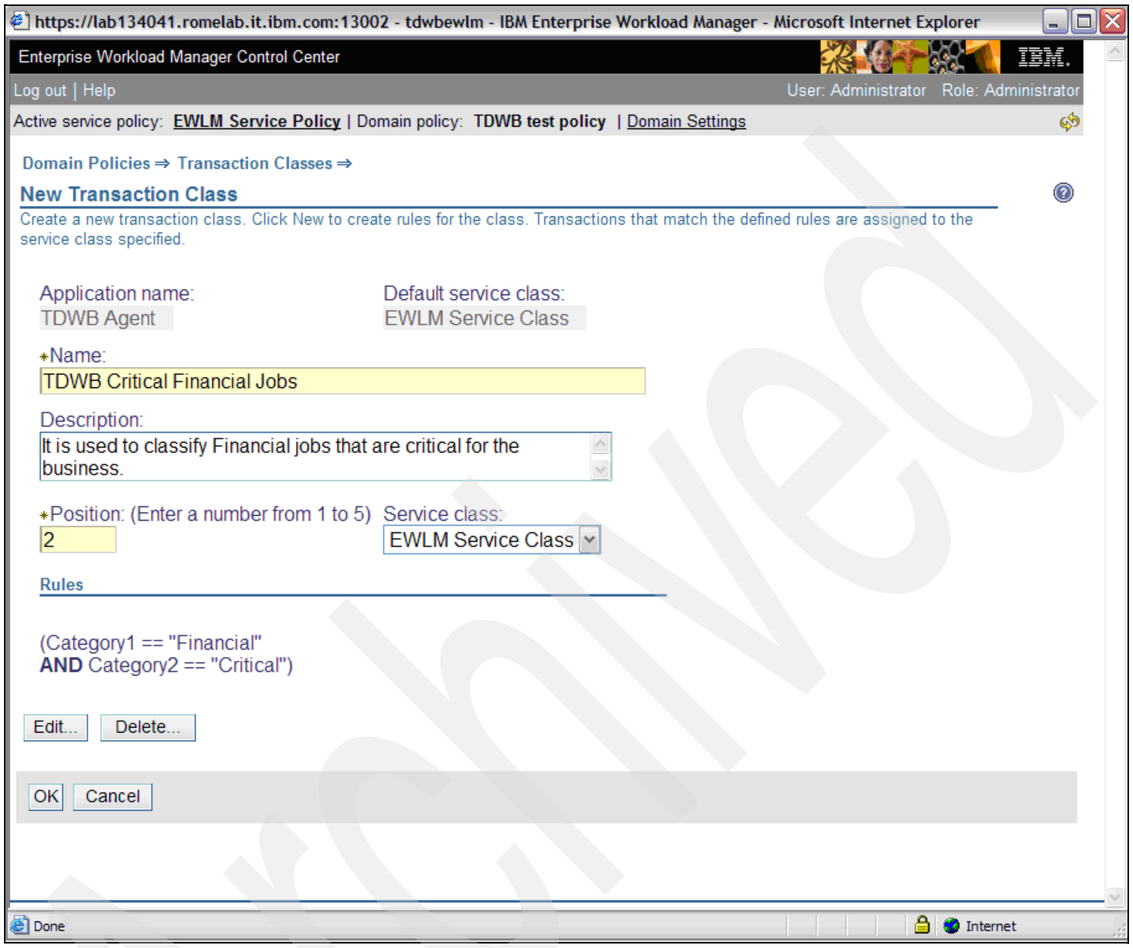
New Transaction Class Rule - 'TDWB Critical Financial Jobs'

Create a rule or set of conditional rules. To create a rule, select a filter type. Select an operation, equal or not equal. Specify a value. Drag the AND and OR operators onto the rule connecting lines to create conditional rules.

Filter Type
- EWLM:Hostname
- EWLM:OS Level
- EWLM:OS Platform
- EWLM:System Name
- EWLM:Transaction name
- Category1
- Category2
- Category3
- JobName

Figure A-18 Creating a rule using multiple Tivoli Dynamic Workload Broker categories

The resulting transaction class is then displayed, as shown in Figure A-19.


https://lab134041.romelab.it.ibm.com:13002 - tdwbewlm - IBM Enterprise Workload Manager - Microsoft Internet Explorer

Enterprise Workload Manager Control Center

Log out | Help — User: Administrator Role: Administrator

Active service policy: **EWLM Service Policy** | Domain policy: **TDWB test policy** | Domain Settings

Domain Policies ⇒ Transaction Classes ⇒

New Transaction Class

Create a new transaction class. Click New to create rules for the class. Transactions that match the defined rules are assigned to the service class specified.

Application name: TDWB Agent

Default service class: EWLM Service Class

*Name: TDWB Critical Financial Jobs

Description: It is used to classify Financial jobs that are critical for the business.

*Position: (Enter a number from 1 to 5) 2

Service class: EWLM Service Class

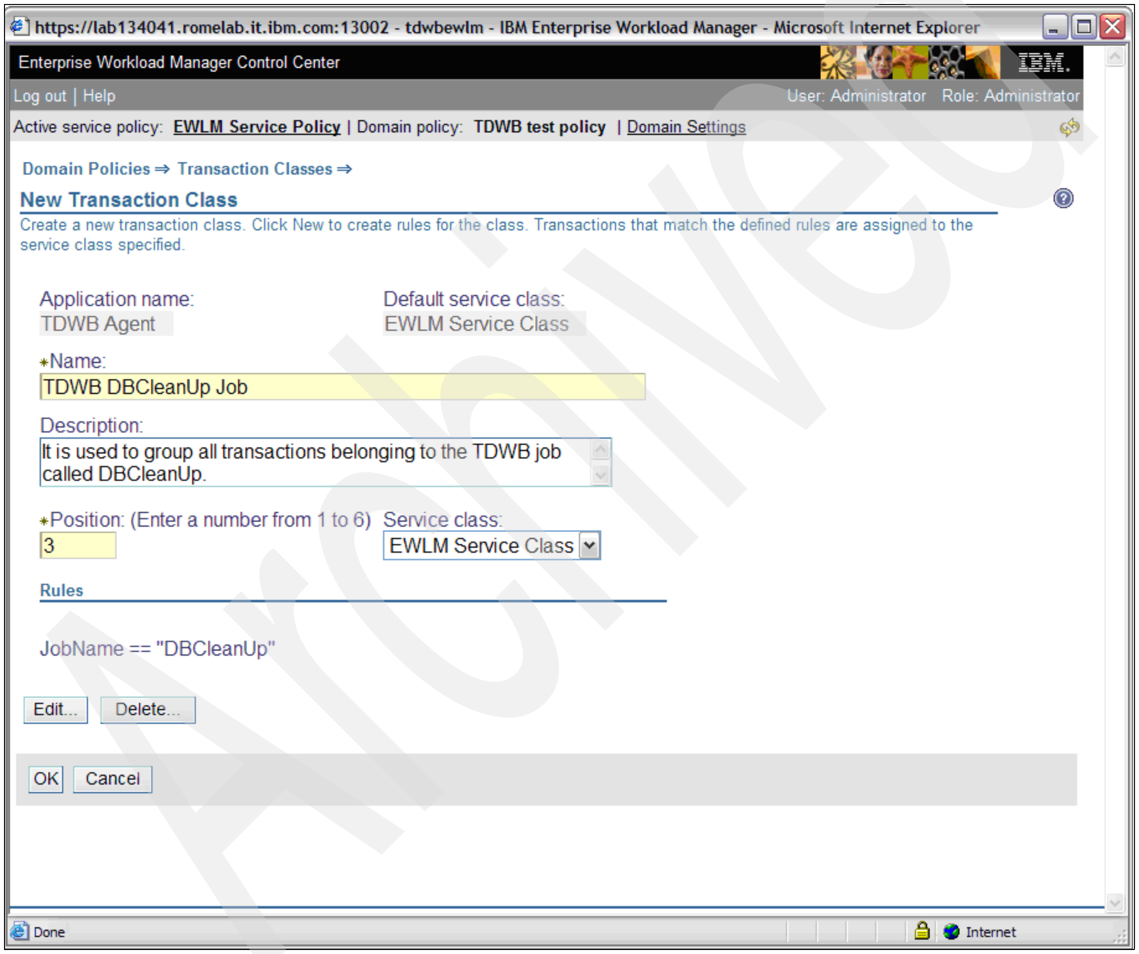
Rules

(Category1 == "Financial"
AND Category2 == "Critical")

Figure A-19 Displaying new multiple category rule in transaction class panel

Confirming interaction from Enterprise Workload Manager Control Center

Enterprise Workload Manager's Control Center displays a list of all load balancers and schedulers that connect to the Domain Manager through SASP. To see the current state of the Tivoli Dynamic Workload Broker connection to Enterprise Workload Manager, simply go to the Load Balancing Panel of the Enterprise Workload Manager Control Center. See Figure A-20.


https://lab134041.romelab.it.ibm.com:13002 - tdwbewlm - IBM Enterprise Workload Manager - Microsoft Internet Explorer
Enterprise Workload Manager Control Center
IBM.
Log out | Help
User: Administrator Role: Administrator

Active service policy: EWLM Service Policy | Domain policy: TDWB test policy | Domain Settings

Domain Policies ⇒ Transaction Classes ⇒

New Transaction Class

Create a new transaction class. Click New to create rules for the class. Transactions that match the defined rules are assigned to the service class specified.

Application name: TDWB Agent

Default service class: EWLM Service Class

*Name: TDWB DBCleanUp Job

Description: It is used to group all transactions belonging to the TDWB job called DBCleanUp.

*Position: (Enter a number from 1 to 6) 3

Service class: EWLM Service Class

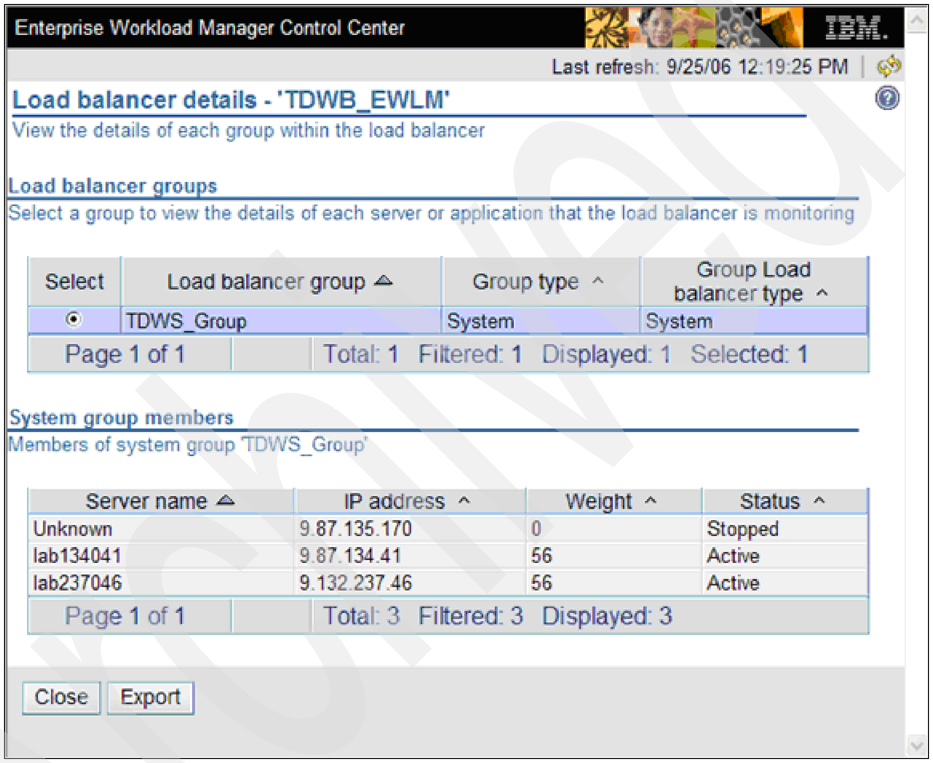
Rules

JobName == "DBCleanUp"

Figure A-20 Enterprise Workload Manager Control Center load balancing panel

Find the Tivoli Dynamic Workload Broker Server in the list identified by its IP address and the load balancing identifier that is the value used for Enterprise Workload Manager Domain Manager Name when installing the Enterprise

Workload Manager plug-in during the Tivoli Dynamic Workload Broker install. (In this case it was Tivoli Dynamic Workload Broker_Enterprise Workload Manager.) Select the **Tivoli Dynamic Workload Broker Server** and the **Details** action to get to the Load Balancer Details panel of the Tivoli Dynamic Workload Broker Server. The Load Balancer Details panel displays all of the groups and group members registered by the Tivoli Dynamic Workload Broker Server. The current state and weight of each group member is also included in the report. An example is provided in Figure A-21.



Last refresh: 9/25/06 12:19:25 PM

Load balancer details - 'TDWB_EWLM'

View the details of each group within the load balancer

Load balancer groups

Select a group to view the details of each server or application that the load balancer is monitoring

Select	Load balancer group	Group type	Group Load balancer type
●	TDWS_Group	System	System

Page 1 of 1 Total: 1 Filtered: 1 Displayed: 1 Selected: 1

System group members

Members of system group 'TDWS_Group'

Server name	IP address	Weight	Status
Unknown	9.87.135.170	0	Stopped
lab134041	9.87.134.41	56	Active
lab237046	9.132.237.46	56	Active

Page 1 of 1 Total: 3 Filtered: 3 Displayed: 3

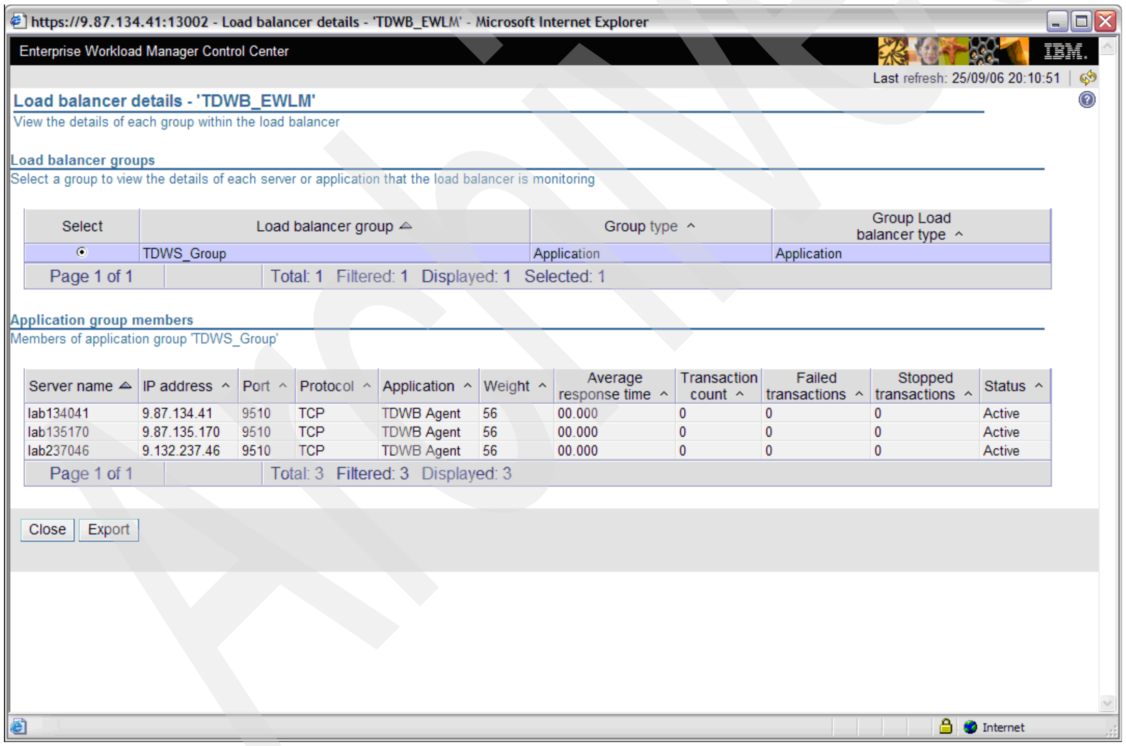
Figure A-21 Load balancer details

Application level load balancing

Enterprise Workload Manager's load balancing algorithm relies on ARM instrumentation to get application information to use when generating load balancing weights (called application level load balancing). If any member in the registered group has not reported ARM statistics, only system statistics will be used when computing weights for the entire group (called system level load balancing). While system load balancing weights are still effective, factors such as application-level failures cannot be factored into the calculation unless the group is at the application load balancing level.

An administrator can determine the load balancing level of each group in the Enterprise Workload Manager Control Center's load balancing panel by looking at the Group Load Balancer Type field of the group. If the Group Load Balancer Type has a value of *System*, the following actions can be taken to change it to *Application*.

1. Make certain that **Application** was chosen for the Weight Scope when installing Tivoli Dynamic Workload Broker and that ARM is enabled on the TDWB Agent (see "Enabling ARM on the Tivoli Dynamic Workload Broker agent" on page 636).
2. Forcefully send a transaction to each member in the group to ensure that the corresponding Enterprise Workload Manager Managed Servers have reported ARM statistics to the Domain Manager. See Figure A-22.


https://9.87.134.41:13002 - Load balancer details - 'TDWB_EWLM' - Microsoft Internet Explorer
Enterprise Workload Manager Control Center
IBM.
Last refresh: 25/09/06 20:10:51

Load balancer details - 'TDWB_EWLM'

View the details of each group within the load balancer

Load balancer groups

Select a group to view the details of each server or application that the load balancer is monitoring

Select	Load balancer group	Group type	Group Load balancer type
●	TDWS_Group	Application	Application

Page 1 of 1 Total: 1 Filtered: 1 Displayed: 1 Selected: 1

Application group members

Members of application group 'TDWS_Group'

Server name	IP address	Port	Protocol	Application	Weight	Average response time	Transaction count	Failed transactions	Stopped transactions	Status
lab134041	9.87.134.41	9510	TCP	TDWB Agent	56	00.000	0	0	0	Active
lab135170	9.87.135.170	9510	TCP	TDWB Agent	56	00.000	0	0	0	Active
lab237046	9.132.237.46	9510	TCP	TDWB Agent	56	00.000	0	0	0	Active

Page 1 of 1 Total: 3 Filtered: 3 Displayed: 3

Figure A-22 Application level load balancing

Enterprise Workload Manager resource allocation for meeting job goals

Enterprise Workload Manager possesses several resource management capabilities, including Logical Partition Management and Provisioning. Once work is classified into Enterprise Workload Manager Goals (see “Enterprise Workload Manager classification of Tivoli Dynamic Workload Broker jobs” on page 637), each of the resource management capabilities may be activated to help Tivoli Dynamic Workload Broker work meet its goals. Resource management capabilities are different for each platform, so appropriate Enterprise Workload Manager documentation (based on the platform) should be consulted if resource management is desired. Future versions of Enterprise Workload Manager may have additional resource management capabilities that can also be employed to meet administratively.

B

Default ports used by Tivoli Dynamic Workload Broker

This appendix contains a list of default ports used by Tivoli Dynamic Workload Broker components. We list the default ports used by:

- Tivoli Dynamic Workload Broker server
- DB2 server used by Tivoli Dynamic Workload Broker server
- Integrated Solutions Console
- WebSphere Application Server hosting the Integrated Solutions Console
- Common Agent hosting the Tivoli Dynamic Workload Broker agent

Ports used by Tivoli Dynamic Workload Broker server

In this section we provide tables listing the ports used by Tivoli Dynamic Workload Broker server.

Table B-1 lists the ports on server used for client Web services calls.

Table B-1 Ports on server used for client Web services calls

Port name/description	Default port number
Unsecure communication (HTTP) with clients via Web services	9550
Secure communication (HTTPS) with clients via Web services	9551

Table B-2 lists the ports used by the Tivoli Dynamic Workload Broker server when integrated with other products.

Table B-2 Ports on server used for communication with integrated applications

Port name/description	Default port number
Tivoli Workload Scheduler (TWS) Agent port	31111
Enterprise Workload Manager (EWLM) domain manager port	3860
IBM Change and Configuration Management Database (CCMDB) server port	9530
IBM Tivoli Provisioning Manager (TPM) Server port	8777

Ports used by Agent Manager

In this section we list the ports used by Agent Manager. Agent Manager is the server side of Common Agent Services. Agent Manager can be installed either on the same machine as the Tivoli Dynamic Workload Broker server or on another machine.

Table B-3 lists the ports used by Agent Manager.

Table B-3 Ports used by Common Agent Services - server side

Port name/description	Default port number
Agent Manager Registration Port. This port uses server-side authentication.	9511
Agent Manager Secure Port. The port number for secure communications with client authentication with mutual authentication.	9512
Agent Manager Public Port. The port number for public communication, including the alternate port for the agent recovery service.	9513

Ports used by DB2 server

In this section we provide a table listing the ports used by a DB2 server used for communication with Tivoli Dynamic Workload Broker server.

Table B-4 Ports used DB2 server

Port name/description	Default port number
DB2 JDBC listening port	50000

Ports used by Integrated Solutions Console

In this section we provide tables listing the ports used by the Integrated Solutions Console.

Ports used by Integrated Solutions Console

Table B-5 lists the ports used by the Integrated Solutions Console, which is the *client* of the Tivoli Dynamic Workload Broker server.

Table B-5 Ports used by Integrated Solutions Console (ISC)

Port name/description	Default port number
Integrated Solutions Console HTTP port	8421
Integrated Solutions Console HTTPS port	8422
Integrated Solutions Console Bootstrap RMI port	8424
Integrated Solutions Console SOAP port	8425
Integrated Solutions Console Admin HTTP port	8431
Integrated Solutions Console Admin HTTPS port	8432
Integrated Solutions Console SAS port	8439
Integrated Solutions Console CSIv2 server authentication port	8440
Integrated Solutions Console CSIv2 mutual authentication port	8441
Infocenter Help port	8423

Ports used by WebSphere Application Server hosting the Integrated Solutions Console

Table B-6 lists the ports used by the WebSphere Application Server hosting the Integrated Solutions Console.

Table B-6 Ports used by WebSphere Application server hosting ISC

Port name/description	Default port number
WebSphere Application Server HTTP port	8426
WebSphere Application Server HTTPS port	8427

Port name/description	Default port number
WebSphere Application Server Bootstrap RMI port	8428
WebSphere Application Server SOAP port	8429
WebSphere Application Server Admin HTTP port	8433
WebSphere Application Server Admin HTTPS port	8434
WebSphere Application Server ORB port	8435
WebSphere Application Server SAS port	8436
WebSphere Application Server CSIv2 server authentication port	8437
WebSphere Application Server CSIv2 mutual authentication port	8438

Ports used by Common Agent

In this section we provide tables listing the ports used by Common Agent. Common Agent is the agent side of the Common Agent Services. Each Tivoli Dynamic Workload Broker agent is hosted by the Common Agent.

Table B-7 lists the ports used by Common Agent.

Table B-7 Ports used by Integrated Solutions Console (ISC)

Port name/description	Default port number
Agent Port. The port number that the common agent uses for communication.	9510
Nonstop service port 1. The port number used by the non-stop server to listen for the common agent process.	9514
Nonstop service port 2. The port number used by the non-stop server to listen for the common agent process.	9515

Port name/description	Default port number
HTTP transport port. The port number used for HTTP transport.	80
HTTPS transport port. The port number used for HTTPS transport.	443

Abbreviations and acronyms

ARM	Application Response Measurement
CAS	Common Agent Services
CCMDB	Change and Configuration Management Database
CIT	Common Inventory Technology
CLI	Command-line interface
CMP	Cluster multi-processing
COM	Common Object Model
CORBA	Common Object Request Broker Architecture
CP	Current Plan
DVIPA	Dynamic Virtual IP Address
E2E	End-to-end
EJB	Enterprise Java Beans
EWLM	Enterprise Workload Manager
FTA	Fault tolerant agent
GUI	Graphical user interface
HA	High availability
HAGS	High Availability Group Services
HATS	High Availability Topology Services
IBM	International Business Machines Corporation
IDEs	Integrated Development Environments
ISC	Integrated Solutions Console
ISPF	Interactive System Productivity Facility
ITSO	International Technical Support Organization
J2EE	Java 2 Enterprise Edition
JBDC	Job Brokering Definition Console
JCL	Job Control Language
JMS	Java Message Services
JSC	Job Scheduling Console
JSDL	Job Submission Description Language
LTP	Long-term plan
MSCS	Microsoft Cluster Service
RA	Resource Advisor
RMC	Resource Monitoring and Control
RMI	Remote Method Invocation
RSCT	Reliable Scalable Cluster Technology
SASP	Server/Application State Protocol
SLA	Service level agreement
SOA	Service-oriented architecture
SOAP	Simple Object Access Protocol
TCA	Tivoli Common Agent
TCO	Total cost of ownership
TDWB	Tivoli Dynamic Workload Broker
TMR	Tivoli Management Region
TPM	Tivoli Provisioning Manager
TSM	Tivoli Storage Manager
TWS	Tivoli Workload Scheduler
USS	UNIX System Services
WLM	Workload Manager
WS	Workstation
WSDL	Web Services Description Language

WSIL	Web Services Inspection Language
WTO	Write-to-operator

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 666. Note that some of the documents referenced here may be available in softcopy only.

- *Deployment Guide Series: IBM Tivoli Provisioning Manager Version 5.1,* SG24-7261
- *Developing Workflows and Automation Packages for IBM Tivoli Intelligent Orchestrator V3.1*, SG24-6057
- *Web Services Handbook for WebSphere Application Server 6.1,* SG24-7257
- *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688

Other publications

These publications are also relevant as further information sources:

- *IBM Tivoli Dynamic Workload Broker User's Guide Version 1.1*, SC32-2281
- *IBM Tivoli Dynamic Workload Broker Installation and Configuration,* SC32-2282
- *IBM Tivoli Workload Scheduler for z/OS Scheduling End-to-end,* SC32-1732
- *IBM Tivoli Workload Scheduler for z/OS Managing the Workload,* SC32-1263
- *IBM Tivoli Monitoring Administrator's Guide Version 6.1.0*, SC32-9408
- *IBM Tivoli Monitoring Installation and Setup Guide Version 6.1.0*, GC32-9407
- *IBM Tivoli System Automation for Multiplatforms Base Component User's Guide*, SC33-8210

Online resources

These Web sites are also relevant as further information sources:

- System prerequisites for DB2 installations on AIX:

  `http://www-1.ibm.com/support/docview.wss?rs=71&uid=swg21181544`
- System prerequisites for DB2 installations on Linux:

  `http://www-306.ibm.com/software/data/db2/linux/validate/platdist82.html`
- System prerequisites for DB2 installations on Windows:

  `http://www-1.ibm.com/support/docview.wss?rs=71&uid=swg21176759`
- Most up-to-date system operating system information for DB2:

  `http://www.ibm.com/software/data/db2/udb/sysreqs.html`
- WebSphere Application Server V6.0.2 hardware requirements summary:

  `http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg27007250`
- WebSphere Application Server V6.0.2 detailed system requirements:

  `http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg27007256`
- Tivoli Dynamic Workload Broker systems requirements:

  `http://ibm.com/support/docview.wss?rs=3190&uid=swg24013539`
- DB2 online manuals:

  `http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp`
- Tivoli Dynamic Workload Broker systems requirements:

  `http://ibm.com/support/docview.wss?rs=3190&uid=swg24013539`
- DB2 online manuals:

  `http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp`

How to get IBM Redbooks

You can search for, view, or download Redbooks, IBM Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM

Redbooks

Getting Started with Tivoli Dynamic Workload Broker Version 1.1

Getting Started with Tivoli Dynamic Workload Broker Version 1.1

Insider's guide to IBM Tivoli Dynamic Workload Broker

High availability and performance considerations

Integration scenarios

IBM Tivoli Dynamic Workload Broker is a key element in a comprehensive, on demand, Tivoli workload automation portfolio. It can use dynamic resource information as well as recommendations from other products to determine the best systems to which new jobs will be dispatched.

This IBM Redbooks publication documents the architecture, installation and customization, operation best practices, performance optimization, high availability considerations, Web Services interface, and troubleshooting of Tivoli Dynamic Workload Broker V1.1.

In addition, we cover integration scenarios with other IBM products, such as IBM Tivoli Workload Scheduler, IBM Tivoli Provisioning Manager, IBM Tivoli Change and Configuration Management Database, IBM Tivoli Monitoring, Tivoli Enterprise Portal, and IBM Enterprise Workload Manager.

Finally, we discuss Tivoli Dynamic Workload Broker operation in a IBM Tivoli Workload Scheduler for a z/OS end-to-end environment.

Clients and Tivoli professionals who are responsible for installing, administering, maintaining, or using IBM Tivoli Dynamic Workload Broker will find this book a major reference.

SG24-7442-00 ISBN 0738489484