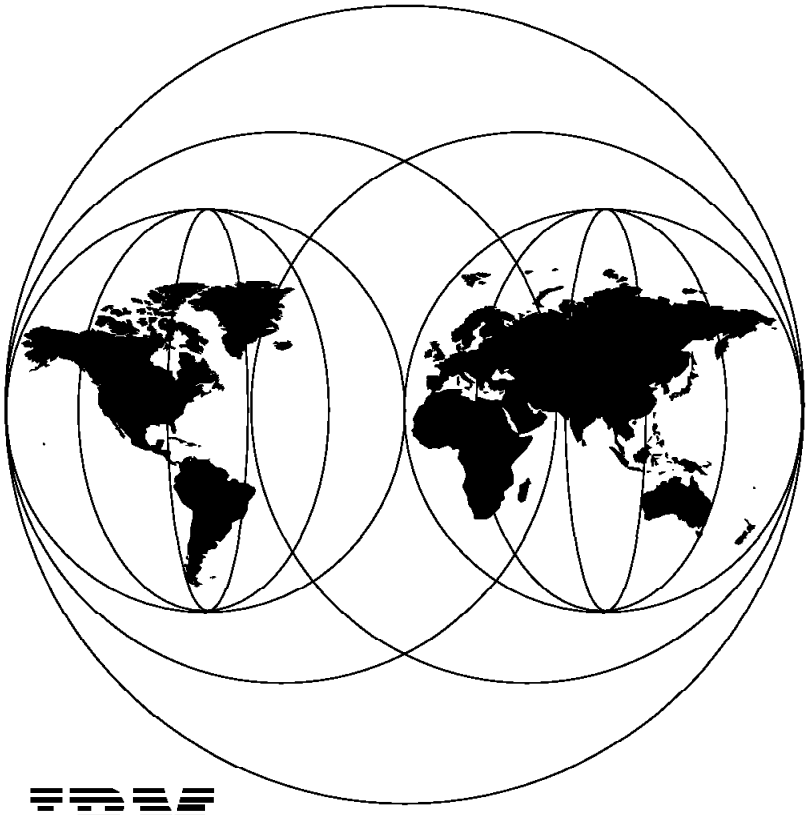
SG24-2071-00

Integrating TME 10 on the RS/6000 SP

September 1997



IBM

International Technical Support Organization
Poughkeepsie Center

International Technical Support Organization

SG24-2071-00

Integrating TME 10 on the RS/6000 SP

September 1997

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix I, “Special Notices” on page 229.

First Edition (September 1997)

This edition applies to PSSP Version 2, Release 2 for use with the AIX Version 4 Operating System and TME 10 Version 3.1.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

This redbook provides practical experiences and suggestions on integrating the IBM Parallel System Support Programs for AIX (PSSP) 2.2 system management and administration tools which run on the RS/6000 Scalable POWERparallel Systems (RS/6000 SP) into the Tivoli Management Environment 10 (TME 10).

This redbook requires indepth knowledge and understanding of TME 10 and intermediate knowledge of the PSSP 2.2 functionality.

This redbook provides help to SP system administrators, SP technical professionals and software engineers to provide them with hints and tips to manage certain PSSP-specific system management aspects in TME 10.

Several practical examples are presented to demonstrate how individual components of TME 10 can be used to interface with PSSP 2.2 specific characteristics.

This redbook covers the following subjects in more detail:

1. Planning for RS/6000 SP and TME 10
2. TME 10 installation considerations
3. SP event management integration with TME 10
4. Task libraries with PSSP commands
5. AEF customizations

Also, attached to this redbook is a diskette containing the code used in this redbook for the PSSP T/EC Adapter code, sample task libraries, and the AEF Customization code. This code can also be found on the World Wide Web at the following URL:

ftp://www.redbooks.ibm.com/redbooks/SG242071.

Knowledge of the following products and disciplines are assumed and not covered in this redbook:

TME 10 product installation

Understanding of the TME 10 products

PSSP 2.2 installation and administration

Understanding of the PSSP 2.2 High Availability Infrastructure

AIX 4 system administration

The Team That Wrote This Redbook

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

Peter Kes is a project manager in the International Technical Support Organization, Poughkeepsie Center. Before joining the ITSO in 1996, Peter worked for the RS/6000 Product Division in the Netherlands as an SP Specialist.

Marcos Novaes is a system engineer in France. He holds a PhD degree in Computer Science and graduated in 1982 from the University of Paris. After his study, he joined the IBM La Gaude Laboratory in France where he was involved in RAS of Communication Controller Products. In 1989 he joined the Toulouse branch office to work in the scientific and technical IBM solutions. In 1996 he moved to Paris to work as a service and support engineer in the ANSS AIX department.

Rosario Uceda-Sosa is the product manager at the Software and Networking brand team in IBM Taiwan. He is responsible for planning, consultant, and technical support for networking software products since 1996. He has six years of experience in networking software. Before joining IBM in 1995, he worked for the CCL/ITRI National Lab for four years. He holds a master degree in computer science at the Illinois Institute of Technology in 1990.

Ron Goering is a RS/6000 IT specialist at the Availability Services department of IBM Taiwan. He is the team leader of RS/6000 professional services team and also responsible for RS/6000 SP support in Taiwan. He is a certified Tivoli consultant and holds nine years of experience in UNIX operating systems. Before joining IBM Taiwan in 1995, he worked for Cray Research Inc., as a field system analyst. His areas of expertise include performance tuning, kernel debugging, parallel programming, and TCP/IP networking.

David Chiu is a system management architect for the RS/6000 SP system. He has been involved in the SP system since its inception.

George Versmissen is a software developer for the RS/6000 SP system with design and development responsibilities in the system management area. She has worked on a number of development projects within IBM over the past 14 years.

Neerav Shah is a software engineer with the RS/6000 division. He is currently involved in the development of the High Availability Infrastructure for the RS/6000 SP system. Before joining IBM in 1995, he was a research scientist at CIS Inc., where he worked in the design of a system for the parallel processing and analysis of sonar signals. He holds a PhD degree from the University of North Texas, and his dissertation proposes a parallel implementation of the Discrete Wavelet Transform for analyzing multidimensional signals.

Eric Chin is a software engineer working for the RS/6000 SP Software Lab since 1995. She holds a PhD degree in Computer Science and Engineering (1995) and a master degree in mathematics (1994) from the University of Michigan. Her areas of expertise are concurrent and distributed systems and OO software architecture. She has been involved in the design of various software projects for the RS/6000 SP system architecture, including the OO framework, and the integration of Tivoli and the SP.

Linda Mellor is member of the Seismic Infrastructure and Support Team at Shell International Exploration and Production B.V, Research and Technical Services, Rijswijk, The Netherlands. His areas of expertise include system management of SP systems and RS/6000 workstations in a seismic processing environment.

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

IBM PPS Lab Poughkeepsie:
Steve Champagne

International Technical Support Organization, Poughkeepsie Center:
Marcello Barios
Endy Chiakpo

International Technical Support Organization, Raleigh Center:
Barry Nusbaum
Dave Shogren

IBM Global Services:
Neerav Shah

Software Levels

The following software levels were used during the development of this redbook. The names and versions are listed as they appeared on the distribution media.

Table 1. Software Levels

Software	**Level**
AIX	Version 4.1.4
PSSP	Version 2.2
Tivoli/Framework	Version 3.1 Revision C
Tivoli/User Administration	Version 3.1 Revision A
Tivoli/Enterprise Console	Version 2.6 Revision B
Tivoli/Sentry	Version 3.0.2 Revision A
TME 10 Distributed Monitoring	Version 2.3.1
TME 10 Software Distribution Manager	Version 3.1
Tivoli/Inventory	Version 3.0 Revision A
NetView/6000 for AIX	Version 4.1.2 PTF U443133

Deliverables

This redbook contains example code and scripts. Most of the examples and scripts are listed in the appendix of this book. A complete copy of the scripts and code is attached to this redbook as a diskette.

Also, a copy will be available on the World Wide Web. The URL for this web site is:

ftp://www.redbooks.ibm.com/redbooks/SG242071

You can download the contents of this directory using an Internet browser.

All code and scripts are provided on an as is basis. Support for the PSSP T/EC Adapter is available through a Programming Request for Price Quotation (PRPQ) from the Poughkeepsie Laboratory.

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in "ITSO Redbook Evaluation" on page 237 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com`
  For IBM Intranet users `http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Chapter 1. Introduction

This book explores how to integrate an RS/6000 SP system with the Tivoli Management Environment. It is intended for customers, system providers, and software engineers who have to perform such an integration.

The Tivoli Management Environment provides a management platform and applications to manage an enterprise computing environment. The RS/6000 SP consists of a number of processing nodes, each running the AIX operating system and connected by an administrative LAN, and an optional high performance switch. The Parallel System Support Program (PSSP) is software that provides administrative and management support for the RS/6000 SP system.

Tivoli applications focus on supporting a broad range of platforms providing a consistent interface to commonly used administrative functions. The main focus of Tivoli applications is managing in the enterprise environment.

The PSSP provides system-specific system management for the RS/6000 SP. The software provided in PSSP provides tools for managing the SP platform, and includes platform-specific management tools. One of the challenges when integrating an SP into a Tivoli environment is to analyze which tasks should be performed through the Tivoli environment, and which should be done using the PSSP tools. Factors such as the frequency of the task, the domain of administration, and the skills of the administrators should be considered.

Figure 1 on page 2 shows the main integration points for integrating SP into the Tivoli environment. These areas are:

- Tivoli applications acting on nodes of the RS/6000 SP system
- Monitoring the state of applications, subsystems, and SP nodes
- Making RS/6000 SP-specific management tasks available to the administrator in the Tivoli environment

There are a number of possible integration scenarios that are discussed more completely in the following chapters. Tivoli applications can be used against individual SP nodes. They treat each node as a standalone RS/6000 system. The full power of Tivoli applications can be used to manage each individual SP node. Tivoli constructs, such as policy regions and profile managers, can be used to group the nodes of the RS/6000 into appropriate groupings. For example, this integration point might be appropriate for managing users using the TME 10 User Administration product.

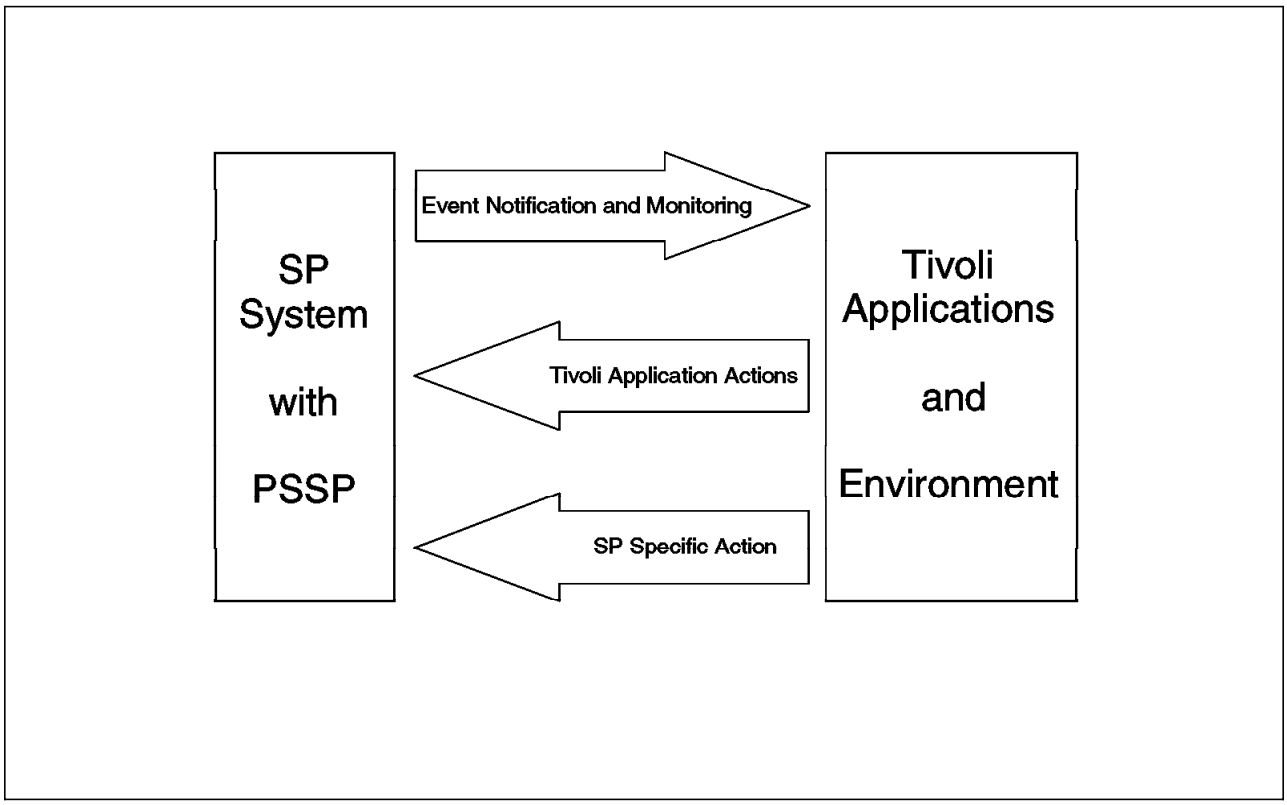


Figure 1. Tivoli and SP integration. Three areas for Tivoli and SP integration are shown.

An important capability of Tivoli is the monitoring of the computing environment. In the most basic scenario, existing TME 10 Distributed Monitoring monitors could be used to monitor resources on individual nodes and forward alerts to TME 10 Distributed Monitoring, and on to the TME 10 Enterprise Console (T/EC) application. A T/EC event adapter has been developed to provide additional monitoring capability on the SP system. It allows any High Availability Infrastructure Event Manager event to be forwarded to T/EC. This allows many SP-unique resources, as well as standard AIX resources, to be efficiently monitored. T/EC can be used as a centralized point for event correlation, notification, and automated actions. More details of the PSSP T/EC Adapter are provided in Chapter 4, “Event Management Integration” on page 25.

The PSSP provides many SP-unique functions, including management of the high performance switch, node control including power controls, and parallel commands. In a Tivoli environment, there are several alternate ways to access these functions.

SMIT or Perspectives could be used to manage the SP directly. Alternatively, a task library consisting of frequently used SP management tasks can be defined in Tivoli. Administrators can execute these tasks against individual nodes of the SP control workstation as appropriate. Sample tasks and more specific information about task library creation is provided in Chapter 5, “Task Libraries, Tasks, and Jobs” on page 95.

The highest degree of integration is provided by extending the TME 10 Desktop so that the managed node objects representing SP nodes provide direct access to additional SP-specific functions. This capability is described in more detail in Chapter 6, “AEF Customizations for the RS/6000 SP” on page 105.

Chapter 2. Planning and Design

Before installing Tivoli on your RS/6000 SP, careful positioning and planning of your administrative tasks, applications, and physical resources is required. This chapter introduces some basic Tivoli concepts, and then discusses positioning your administrative tasks and applications within your Tivoli enterprise and reviews the planning issues required to deploy Tivoli across your resources and configure your enterprise appropriately.

2.1 Basic Tivoli Terminology

This redbook assumes the reader has full knowledge of Tivoli terminology, concepts, and applications. This section provides a brief overview of the major concepts that are discussed in this book. For in-depth discussions of these items and other Tivoli terms not included in this list, refer to the *TME 10 Framework Planning and Installation Guide*, *TME 10 Framework User's Guide*, and individual Tivoli application manuals.

Tivoli Management Environment (**TME**)

The combination of the base TME 10 Framework, the distributed object databases, graphical user interface (GUI), command line interface (CLI), and all Tivoli toolkits and applications required within the enterprise that is managed by Tivoli.

TME 10 Framework

The base set of Tivoli software that is required to run any of the Tivoli management applications. The Framework provides basic system administration capabilities, and services for the management applications, including an administrator facility, scheduler facility, and notice facility. The TME 10 Framework is often referred to as the Tivoli Management Platform (TMP).

Tivoli Management Region (**TMR**)

The basic physical unit of Tivoli functionality. It consists of one TME server and the clients that server is managing.

Policy Regions, Policies

A policy region is a collection of TME resources that are controlled by a common set of policies or rules. Typically, policy regions define the boundaries of the authority of Tivoli administrators, as well as provide a mechanism for organizing and managing system resources in a hierarchical structure.

Profiles, Profile Managers

A profile contains a collection of application-specific information. The information in a profile is specific to a particular profile type (for example, a user profile will contain user names, login ids, and so on).

A profile manager contains profiles and a list of subscribers to which the profile data can be distributed. Subscribers can be managed nodes, PC managed nodes, NIS Domains, and other profile managers.

Administrators

A Tivoli administrator is a user that has been given authorization to perform management tasks in the TME. The administrator's authorization roles determine what tasks that administrator can perform against a particular set of resources.

Resources

A TME resource is a general term used to define systems, devices, services, and facilities in a distributed system. A managed resource can be owned by only one policy region.

TME 10 Application Extension Facility (**AEF**)

An interface to dynamically customize the Tivoli applications by adding site-specific behavior or values to standard applications.

TME 10 Event Integration Facility (**EIF**)

A facility to build event adapters to map events from any application, resource, or component into a format compatible with the TME 10 Enterprise Console.

TME 10 Application Developers Environment (**ADE**)

Programming tools for creating new custom management applications on top of the TME 10 Framework.

2.2 Planning the Integration of Your SP with Tivoli

To integrate an SP system into a Tivoli environment, there is interaction and overlap between TME and the PSSP in several areas that must be considered. In general, you must examine the domain of administration for different tasks, and pick the technology that is appropriate for the task. For example, if users are being managed across the SP, other servers, and client machines, then Tivoli user management would be an appropriate choice. If a Tivoli application is chosen to handle a particular task or requirement, you must also design how it integrates with the SP. Among the areas to consider are:

- Security
- System Installation and Maintenance
- File Distribution
- User Management and Administration
- System Monitoring and Event Handling
- Distributed Task and Command Execution

When planning for SP and Tivoli integration in various areas, you should consider the following factors:

- What are the administrative tasks frequently performed in your environment?
- Who performs these tasks and what platform are they familiar with?
- Is the scope of a particular task limited to the SP platform, or does it operate across many platforms in the environment?

Tasks that are performed across many platforms or need centralized data, such as user administration, are good candidates to perform through Tivoli. Infrequently performed, platform-specific tasks, such as setting the switch clocking parameters, probably make sense to have accessible only through platform-specific management tools such as the SP GUI Perspectives, or AIX management tools such as SMIT. Tasks that are platform-specific, but which

may be needed during general operations by personnel who are not specifically trained on platform-specific interfaces are good candidates for integration through Tivoli task libraries, or extensions to the TME 10 Desktop.

2.2.1 Security

Tivoli and SP administration have completely separate and unrelated security domains. SP uses Kerberos V4 as the basis of authentication and authorization of administrative users. Tivoli has several possible security implementations including Kerberos as an option. Even in an environment where both Tivoli and SP are using Kerberos, it is not possible to use only a single Kerberos database. Tivoli and SP security realms must be administered separately.

One possible advantage of Tivoli/SP integration is that operations personnel could be enabled through Tivoli tasks, or Tivoli applications, to perform operations on the SP without being in the SP Kerberos database. Tivoli's authorization structure for tasks using the super, senior, admin, and user roles is easier to manage than the SP mechanisms of having separate access control lists (ACLs) for each resource.

2.2.2 System Installation and Maintenance

The PSSP provides facilities for the initial installation of the operating system, AIX, and the basic PSSP software. It exploits the AIX Network Installation Manager (NIM) to provide this capability. Tivoli does not have equivalent function, and it is recommended that the PSSP and AIX facilities be used for SP installation, migration, PTF application, and other software maintenance.

There are also many platform-specific administrative tasks that Tivoli does not provide. Generally these tasks are infrequently performed, and are quite operating system- or hardware-specific. We generally recommend that they be performed by experienced personnel using SMIT, Perspectives, or the command line. However, Tivoli tasks could be defined for the more common of these tasks, and the Tivoli role structure used to allow operators to perform these tasks from the TME 10 Desktop. For more details on invoking SP-specific tasks from Tivoli, see Chapter 5, “Task Libraries, Tasks, and Jobs” on page 95. TME 10 Desktop customizations could be used to allow the launching of SMIT or Perspectives, or running a specific command. See Chapter 6, “AEF Customizations for the RS/6000 SP” on page 105.

2.2.3 File Distribution

Both the SP and Tivoli provide mechanisms for distribution and update of groups of files. These files could be application binaries, application data, or system files.

The SP provides the file collections technology. It is specific to the SP, and allows groups of related files to be defined into file collections. Files which have been changed in a file collection are automatically updated periodically, or when a node reboots. It is reasonably easy to add new files to an existing file collection, although there is no SMIT or GUI interface to do so. Defining a new file collection can be difficult.

The SP predefines file collections used for user management files (/etc/passwd, /etc/group, /etc/security/*, and related files), and another for system files such as /etc/services.

TME 10 Software Distribution is a Tivoli application that is appropriate when file synchronization is managed at a domain larger than the SP. It provides a rich set of facilities for distributing and updating files. TME 10 Software Distribution operates against individual SP nodes exactly as it would against any other managed nodes. Profile managers and policy regions can be used to group SP nodes to make file distribution management easier.

2.2.4 User Management

Both Tivoli and the PSSP provide tools for user administration. The PSSP solution provides a simple mechanism for managing users only on the SP platform, and optionally integrates support for automounting user's home directories. TME 10 User Administration is an enterprise user administration application that can provide user management across many platforms with configurable levels of local control or centralization.

The decision on which user management solution to deploy depends primarily on the scope of user administration in your enterprise. Among the reasons you might want to choose TME 10 User Administration are:

- To enforce a common userid space for a domain larger than the SP
- To manage different sets of users on different SP nodes
- To provide different administrators with administrative control on different nodes or sets of nodes
- To integrate your SP user management into the enterprise
- To eliminate the SP restriction that requires users to change their passwords on the control workstation

TME 10 User Administration allows you to manage your users at the enterprise level, or across a number of machines, something that PSSP management by itself clearly does not do. Using the `wpasswd` command, you can have your users change their passwords from any SP node, and TME 10 User Administration will keep those passwords synchronized across SP nodes and other systems. By using profile managers you can enforce a common userid space, and restrict access of certain users to certain nodes. You can also allow administrators local control over users on a node or group of nodes while enforcing common policies at the enterprise level. TME 10 User Administration does not handle building automounter maps as SP user management does.

2.2.5 System Monitoring and Event Handling

Monitoring key system resources is probably the first and most important integration point to be considered. It is also the area where there are the most choices for a solution.

The first task is to identify the key resources you wish to monitor. These resources might include node availability, adapter or network availability, state of file systems, processor load, and other states of computing resources, as well as the state of key applications.

There are a number of monitors available including the following:

- TME 10 Distributed Monitoring provides simple monitoring for common computing resources. TME 10 Distributed Monitoring alerts can be forwarded to TME 10 Enterprise Console, or converted into SNMP traps and forwarded to NetView/6000 for AIX.

- TME 10 Enterprise Console logfile adapter can monitor SP logs and be configured to generate events from log entries.
- Several solutions for using Event Management and Problem Management from the PSSP software exist. These solutions involve invoking a program through Problem Management as a response to an Event Management detected event. Using the SP event manager allows a large number of SP-specific subsystems, as well as basic AIX resources, to be efficiently monitored. This combination can be set up using the SP Perspectives GUI. These Problem Management actions in response to an event include:
  - Using the Tivoli `wasync` command to forward a string to T/EC
  - Generating an SNMP trap that is forwarded to NetView
  - Invoking the new tecad_pssp which will parse the event manager event and forward it to T/EC for handling

You should also identify the destination of any monitoring or alerts and the transport for the events. The primary event transport mechanisms are T/EC events, TME 10 Distributed Monitoring monitoring, or SNMP traps. Possible event destinations might be the Tivoli Enterprise Console (T/EC), TME 10 Distributed Monitoring application on a TME 10 Desktop, or NetView or other network manager. The combination of resources to be monitored, and the destination of the monitoring, will help you choose an appropriate monitoring solution. Chapter 4, “Event Management Integration” on page 25 provides detailed discussions of monitoring integration choices and scenarios.

2.2.6 Distributed Task and Command Execution

Both the PSSP and Tivoli provide capabilities for distributed task and command execution. The PSSP provides `dsh`, built on `rsh` with enhancements for parallel execution and formatting, and `sysctl`, a generalized remote task execution backend.

`sysctl` allows adminstrators to define procedures written in any language (Perl, ksh, or C are popular choices), and give selected users access to that procedure through access control lists. The defined procedure executes as root, but the user never needs to know the root password. This capability allows administrators to delegate portions of root access to users without compromising the root password. The authentication for sysctl procedures is based on Kerberos, and users of these procedures must be defined in the Kerberos database, and currently have Kerberos credentials (typically acquired by the `kinit` command.)

Tivoli provides a similar capability using the Tivoli tasks and a Tivoli task library. Tivoli tasks are scripts defined to Tivoli to perform specific tasks on the target host. The execution environment (userid, groupid, working directory, and so on) are specified during task definition. A role is given to a task such that a user must have that role to execute the task.

The main decision point is probably the overall administrative environment. If your administrators are primarily using Tivoli, defining a Tivoli task is the logical way to proceed. Your security environment is also a consideration. `sysctl` requires the user to be defined in the SP Kerberos database. `sysctl` and `dsh` are more efficient mechanisms for executing tasks directed specifically at SP nodes. Chapter 5, “Task Libraries, Tasks, and Jobs” on page 95 provides

additional detail on a number of candidates for Tivoli tasks to help manage the SP.

A hybrid approach that uses dsh or sysctl as commands within a Tivoli task is also possible. This approach combines the benefits of the efficient task execution environment on the SP with the centralized management provided by Tivoli.

Chapter 6, “AEF Customizations for the RS/6000 SP” on page 105 describes TME 10 Desktop customizations that allow execution of a command on a node or set of nodes. These customizations can also be used to provide direct access from the TME 10 Desktop to SP-specific tasks such as node and switch control.

2.3 Planning the Deployment of Tivoli on Your RS/6000 SP

A computing enterprise can include many different types of machines, operating systems, applications, and communication networks. To manage these resources, a Tivoli Management Environment may contain any number of physical TMRs, logical policy regions with possibly many levels of hierarchy, and managed resources which are to be controlled from Tivoli. Tivoli provides you with the framework to manage everything in a consistent manner.

If your operations environment contains one or more RS/6000 SP machines, you may wish to manage these machines as part of your larger Tivoli enterprise. Before introducing your RS/6000 SP into your Tivoli enterprise, there are numerous deployment planning issues that must first be considered. Many of these issues are general considerations that apply to any resources added to a Tivoli enterprise. These considerations will not be discussed in this book since they are fully covered in Tivoli publications and other redbooks.

This chapter will focus on the planning issues that are unique to dealing with an RS/6000 SP in a Tivoli enterprise.

2.3.1 TMR Configuration

The *TME 10 Framework Planning and Installation Guide* contains detailed information on planning your Tivoli Management Region (TMR) configuration. It discusses issues such as:

- If and when you need multiple TMRs
- The location of TME 10 databases and whether to share library and binary files
- Resource updates between TMRs
- Server load
- Number of clients
- Network limitations
- Local versus central administration
- Network communications

Our redbook assumes that the reader is completely familiar with the concepts in this area. However, we will address some considerations specific to the RS/6000 SP.

2.3.1.1 Representing an RS/6000 SP in TMRs

A Tivoli Management Region (TMR) is a physical representation consisting of a TME server and some number of TME clients. There are many ways to represent your RS/6000 SP within TMRs:

- As a single TMR

  The most obvious approach is to represent the entire RS/6000 SP in its own TMR. The TME server would be located on the control workstation or one of the SP nodes, and all other nodes would be TME clients. This provides a simple mapping of your TMR to your physical environment for moderately sized RS/6000 SPs. However, if your RS/6000 SP has a large number of nodes, or if the Tivoli installation stresses other server load characteristics, you may need to divide the RS/6000 SP into multiple TMRs to reduce the impact to a single TME server.

- As a subset of a TMR

  The RS/6000 SP may be represented as a collection of TME clients within a larger TMR that contains other workstations or RS/6000 SPs. This is especially useful if you have smaller RS/6000 SPs. You may also wish to include related print servers, file servers, and other resources that are used during the operation of the RS/6000 SP(s). The TME server may or may not reside within the RS/6000 SP. This is not a feasible representation for larger RS/6000 SPs, where the number of nodes would exceed the TME server capacity.

- As multiple TMRs

  Due to the size of your RS/6000 SP or other physical or organizational considerations, you may wish to represent the RS/6000 SP within Tivoli using multiple TMRs. The location of the TME servers and the distribution of the nodes among the TMRs must be planned. If your RS/6000 SP has multiple partitions, you may consider separating your TMRs on partition boundaries. When dealing with multiple TMRs, it is important to remember that not all TME resources are shared across connected TMRs.

- Only represent the control workstation within a TMR

  If you want to administer and monitor the RS/6000 SP through the control workstation, you may wish to represent only the control workstation in an existing TMR and not represent the nodes at all. With this approach, you will not be able to apply special administration to individual nodes and you may need to develop customized interfaces to meet your needs. However, this approach may greatly reduce the complexity of your Tivoli installation and limit the impact Tivoli has on your SP.

- As a collection of RS/6000 workstations

  Although other configurations will most likely be more appropriate, you could simply represent your RS/6000 SP as an unrelated collection of RS/6000 workstations with no regard to the fact that they are part of an RS/6000 SP. This method does not require any special SP considerations, and you would follow all documented Tivoli procedures for planning this approach. You can have your TME servers reside anywhere in your installation, and have the SP control workstation and nodes be members of whatever TMRs are appropriate. Note, however, that you may not be able to do specific SP administration from Tivoli when managing your environment in this way.

- Have no direct representation within a TMR

You may choose not to represent your RS/6000 SP within a TMR at all. With this approach, you would not install Tivoli on the control workstation or any nodes and you would not be able to manage the RS/6000 SP directly from Tivoli. However, you would still be able to monitor events on the RS/6000 SP through a TME 10 Enterprise Console installed on an external system using an unsecure communications channel to the RS/6000 SP.

The particular representation that you choose for your RS/6000 SP is dependent on many factors, such as the number of nodes on the RS/6000 SP, its organizational structure and use, communication links, size of TME servers, TMR connections, and so on. You must be familiar with the details discussed in the *TME 10 Framework Planning and Installation Guide* to plan the TMR representation of your RS/6000 SP.

2.3.1.2 Communication Considerations

TME distributed architecture is designed to work across a wide variety of LANs and network topologies. The TME server and TME clients within a TMR must communicate with each other. TME servers in connected TMRs must also be able to exchange information and services. The minimal Tivoli requirement is for a bi-directional, full-time, interactive TCP/IP line. The amount of Tivoli-generated network traffic is determined by the type of administration, data distribution, and resource monitoring you plan on performing within your Tivoli installation.

The system topology of your RS/6000 SP with regard to the SP Ethernet, SP switch (if any), outside network connections, routers, and gateways must be carefully considered when planning the location of the TME servers and clients. You must also consider the current traffic on these connections and the impact additional Tivoli communications will have on them. See the *TME 10 Framework Planning and Installation Guide* for a discussion of network limitations and how they affect your TMR configuration.

The SP switch provides a high performance communication link which is unique to the RS/6000 SP. Installations typically reserve this communication path for production applications and critical data access. For most installations, use of the SP switch is not recommended for distributing Tivoli administrative data when other paths exist that would provide adequate network throughput. However, you may find that your production SP environment can tolerate the additional overhead of the Tivoli communications on the SP switch. In this case you will see improved Tivoli performance by installing and managing your TMR across the switch. 3.1, “Installing Tivoli Using the SP Switch” on page 19 contains instructions on how you can do this.

2.3.1.3 TME Server Load

TME server load is one of the most critical factors in determining the size of a TMR. The server load is dependent on the number of file descriptors needed to maintain contact with clients, the network traffic, and the CPU and memory demands Tivoli places on the server.

The server load and network connectivity will also determine where you decide to locate the TME server with regard to your RS/6000 SP. The possible locations are:

- On the SP control workstation

If the TME server is to be located on the control workstation, you must consider the impact TME operations will have on the performance of other control workstation operations that are critical to the functioning of your RS/6000 SP.

- On an SP node

  There are no special SP considerations when placing the TME server on an SP node other than the impact to non-Tivoli processing planned for that node. Appropriate network connections and IP connectivity to all managed nodes must exist. These connections should not be limited by SP partition boundaries.

- On an independent RS/6000 workstation

  Locating the TME server on an independent RS/6000 workstation that is not part of your RS/6000 SP will eliminate the server load impact to your RS/6000 SP. This is only possible if you have an external network connection to your RS/6000 SP and its nodes.

2.3.1.4 Number of TME Clients

The number of remote connections within a single TMR is limited to approximately 200 TME clients. The TME server maintains an open connection with each client and thus is limited by the number of file descriptors that may be open at one time. Realistically, the server load and network impacts will most likely limit the number of clients in your TMR long before this limit of 200 clients is reached.

This limitation on the number of clients impacts how you can represent your RS/6000 SP within a TMR. Smaller RS/6000 SPs with one or two frames can be contained within a TMR. However, larger RS/6000 SPs with multiple frames and large numbers of nodes may need to be divided into multiple TMRs. The actual number of SP nodes that can be supported within a single TMR is dependent on the server load issues discussed in the previous section with relation to the size of the processor chosen for the server.

2.3.1.5 TMR Connections

Individual TMRs can be linked together with either one-way or two-way connections, allowing administrators to manage resources in other TMRs. Restrictions exist as to the types of resources that can be shared across TMRs. See the *TME 10 Framework Planning and Installation Guide* for discussions on connecting TMRs and updating resources between them.

There are no special SP considerations when connecting TMRs. However, you will want to consider the limitations of performing administrative activities across connected TMRs when you decide whether your RS/6000 SP should be contained within a single TMR or divided into multiple TMRs. Some important resources that are not shared across TMRs are the user and group name databases and group profiles. You may, however, overcome some of these limitations through careful implementation and administration of profiles and profile managers which can span TMRs.

2.3.1.6 Location of TME 10 Files

The *TME 10 Framework Planning and Installation Guide* discusses what to consider when deciding where to store the TME 10 databases and library and binary files. There are no special considerations for the RS/6000 SP. To reduce the storage requirements on individual nodes, you can use a common data repository such as NFS or some other shared file system to contain the Tivoli library and binary files.

2.3.1.7 Sizing Considerations

Processor, memory, and disk space recommendations for the TME server and clients are detailed in the *TME 10 Framework Release Notes*. When reviewing these recommendations in relation to the RS/6000 SP, consider these recommendations as additional requirements to those required by the RS/6000 SP that are specified in the *IBM RS/6000 SP Systems Planning Guide*.

2.3.1.8 Application Considerations

Each Tivoli application has its own planning issues that must be considered. For example, if you are installing TME 10 Enterprise Console and/or NetView/6000 for AIX to monitor and respond to problems on your RS/6000 SP, you will need to determine where those servers should reside, their server loads, network connections, and so on. Review the documentation for each application to determine if there are additional requirements to your overall Tivoli plan.

2.3.2 TME 10 Framework Configuration

While the TMR planning focuses on the physical configuration issues, the TME 10 Framework planning deals with the logical configuration of the Tivoli enterprise. The TME 10 Framework configuration should be developed to match your current organizational and administrative structure. The TME 10 Framework provides the basis for managing resources in a distributed environment. The planning considerations for the TME 10 Framework configuration include:

- Policy regions and subregions
- Managed nodes
- Profile managers and subscribers
- Administrator roles

The basic discussions for planning your TME 10 Framework configuration is covered in the *TME 10 Framework User's Guide*. Some SP-specific considerations are included in the following sections.

2.3.2.1 Policy Regions and Subregions

Policy regions are used to organize the resources managed in your Tivoli environment. A policy region is not limited to the local TMR -- it can contain managed resources from connected TMRs. Therefore, you can eliminate some of the physical boundaries that may have been imposed with TME server limitations by using policy regions to generate a logical view of the organizational structure of your resources.

A managed resource such as a managed node may only be resident within a single policy region. However, you may have a hierarchy of policy sub-regions to match the hierarchical view of your resource administration.

There are many ways to organize the managed nodes representing your RS/6000 SP control workstation and nodes in policy regions. A few possibilities may include (but are certainly not limited to) any combinations of the following:

- A top-level policy region for the entire RS/6000 SP

  Typically, there is one governing policy region for all the managed resources within a TMR. If your RS/6000 SP is represented by a single TMR or a subset of a larger TMR, and you have a simple administrative organization, a single policy region containing the control workstation and all of the nodes of your RS/6000 SP as managed nodes may be the simplest approach.

  If your RS/6000 SP spans multiple TMRs, you may find it useful to have a higher-level policy region that includes all of the managed nodes from all of the TMRs representing the SP. This policy region may either contain the managed nodes directly, or indirectly through nested policy sub-regions. This top-level policy region could then be used for generic management of the entire RS/6000 SP, consistent monitoring and administration across all of the SP nodes, and so on.

- Policy sub-regions to match SP partitions

  If your RS/6000 SP contains multiple partitions, you may find it useful to represent the nodes of a partition within separate policy sub-regions. This will allow you to manage the partitions independently. These sub-regions may or may not be hierarchically part of a governing policy region that represents the entire RS/6000 SP.

  An alternative way to represent partitions would be through profile managers. See 2.3.2.3, “Profile Managers and Subscribers” on page 14.

- Policy sub-regions to match disjoint groups of SP nodes

  If you have many SP nodes that are similar to each other or used in a common manner, you may wish to organize such a group into its own policy sub-region for common administration. For example, in a LAN-consolidation environment, you may wish to group the SP nodes into disjoint sets based on organizational or operational characteristics and represent each set as a policy sub-region. As with partitions, these sub-regions may or may not be hierarchically part of a governing policy region that represents the entire RS/6000 SP.

  Using policy sub-regions for groups of SP nodes is most appropriate when the grouping will remain stable over time. For more dynamic grouping, or as another grouping alternative, profile managers can be used.

2.3.2.2 Managed Nodes

Each SP control workstation and node that is installed as a TME server or client is logically represented as a managed node within one (and only one) policy region. You may wish to customize these managed nodes to provide unique SP functions to your administrators from within the TME 10 Desktop. You can develop customizations using Tivoli facilities such as ADE or AEF. See Chapter 6, “AEF Customizations for the RS/6000 SP” on page 105 for information on how you might provide your own TME 10 Desktop customizations and for examples of modifying your managed nodes to invoke functions unique to an SP control workstation and nodes. Also, you can load a set of sample TME 10 Desktop customizations from the diskette provided with this redbook. See 6.2, “Installation of the Sample Customizations” on page 113 for installation instructions.

2.3.2.3 Profile Managers and Subscribers

Profile managers contain profiles to manage the administrative data on your systems. More importantly, however, they contain a list of subscribers that can either be managed nodes or other profile managers. These subscription lists can be a useful organizational construct in representing groups of resources requiring common administration, especially since a managed node or profile manager may be included in any number of subscription lists.

With regard to the RS/6000 SP, you will probably find it useful to create profile managers to represent the following:

- All nodes on your RS/6000 SP
- The control workstation and all nodes on your RS/6000 SP
- All nodes within an SP partition
- All nodes within an SP node group
- All nodes with a common set of operational or organizational characteristics

These profile managers can then become subscribers for task execution, data distribution, profile distribution, and so on. In SP terms, a profile manager that contains only a subscription list can be thought of as an SP node group. Therefore, any SP node groups that have been created for common administrative purposes would be a good candidate for a profile manager.

2.3.2.4 Administrator Roles

By assigning appropriate authorizations and copying and moving policy regions to the correct TME 10 Desktops, you can control the administrative roles of the administrators managing subsets of the Tivoli enterprise. With regard to the RS/6000 SP, you can restrict SP resources and tasks to only those administrators that will be managing the RS/6000 SP from Tivoli. You would do this using the normal procedures for configuring TME 10 administrators as described in the *TME 10 Framework User's Guide*.

2.3.3 Planning for RS/6000 SP Changes

Once a RS/6000 SP has become part of a Tivoli enterprise, planning RS/6000 SP changes also requires planning corresponding TME changes. Some examples of RS/6000 SP changes that may affect your Tivoli installation include:

- Adding new nodes to your RS/6000 SP

  After you have added new nodes to your RS/6000 SP and have verified that they are operating properly within your system, you will want to include them in your Tivoli Management Environment. This will require you to install all of the necessary TME framework and application binaries, libraries, and databases to create the new managed nodes. You will need to add these resources to the appropriate policy regions and profile managers and any specific application resource managers to make them a part of your managed environment.

  If the number of new nodes is significant, you may need to consider creating a new TMR if the new nodes will exceed the limits of your current TME server or reduce its performance to unacceptable levels. You will then need to evaluate how to connect this new TMR to your current Tivoli Management Regions, and how it will be administered within your organization.

- Removing nodes from your RS/6000 SP

Before you remove nodes from your RS/6000 SP, you will want to delete the managed nodes from your TMR and from all policy regions and resource managers within your Tivoli enterprise.

- Re-partitioning your RS/6000 SP

  If you have made TMR and TME 10 Framework configuration decisions based on the system partition layout of your RS/6000 SP, you will need to consider the impact to these configurations if you change the SP partitioning. You may need to move managed nodes from one TMR to another (which requires deleting them from the old TMR and creating them in the new TMR), move the managed nodes from one policy region to another, or change the subscription lists in various profile managers to reflect the new organization.

- Changing IP address and host names

  If you change the IP address or host name for the SP nodes or the control workstation that the TME uses to communicate with, you will need to update the TME server and client databases.

- Migrating your RS/6000 SP to new levels of AIX or PSSP

  When planning to migrate your RS/6000 SP to a new level of AIX or PSSP, you will need to consider how you will migrate your Tivoli installation on those managed nodes. If your TME server is resident on the control workstation or one of the SP nodes, you will have additional migration issues to consider. You will need to decide if you must migrate to new levels of the TME 10 Framework or any other TME applications. You will need to preserve all databases, binaries, libraries, and other files associated with the TME 10 Framework and applications.

Tivoli procedures for all of these changes are described in the *TME 10 Framework Planning and Installation Guide*, the *TME 10 Framework User's Guide*, the *TME 10 Framework Release Notes*, and the individual TME application documents.

2.4 Planning Applied to a Practical Example

To illustrate some of the planning issues that have been discussed in this chapter, we will take a sample SP environment and describe some of the planning choices that can be made to incorporate it into a larger Tivoli enterprise. The example is for discussion purposes only and is not meant to represent any existing Tivoli enterprise or corporate environment.

2.4.1 Environment at the ITSO SP Lab

This redbook was written based upon our experiences while working at the International Technical Support Organization (ITSO) Poughkeepsie Center. Therefore, we will use the ITSO lab as the example environment that we want to manage. The following hardware was available at the ITSO lab for our work:

- A 16-node RS/6000 SP containing:
  - One frame
  - 16 thin nodes
  - A High Performance Switch
  - An SP Ethernet connection to all nodes
  - A control workstation

This RS/6000 SP is referred to as SP2, with the control workstation hostname sp2cw0, and node hostnames sp2n01 through sp2n16 for Ethernet access and sp2sw01 through sp2sw16 for switch access. There is only one partition on this RS/6000 SP.

- A 12-node RS/6000 SP containing:
  - One frame
  - 4 thin nodes
  - 4 wide nodes
  - 1 SMP node
  - An SP Switch
  - An SP Ethernet connection to all nodes
  - A control workstation

  This RS/6000 SP is referred to as SP21, with the control workstation hostname sp21cw0, and node hostnames sp21n01 through sp21n16 for Ethernet access and sp21sw01 through sp21sw16 for switch access. There is only one partition on this RS/6000 SP.
- Several RS/6000 workstations
- Local Token Ring connection between the control workstations and the RS/6000 workstations

2.4.2 Integration Planning

We will make the assumption that administration of the lab machines is self-contained with very little interaction with a more global corporate administrative organization. However, to reduce personnel costs, we will say that the administrators in charge of this lab are also responsible for other labs within the region. The entire region is using Tivoli to manage their systems. The administrators responsible for the lab have complete management control over all machines.

Since the administrators must manage several labs, they would prefer to handle as much SP system management from their TME 10 Desktops as possible. They will install all SP customizations provided with this redbook, and create their own task libraries and tasks for commonly performed operations not included in the customizations.

The users in the lab require access to all available SP nodes and RS/6000 workstations, and would like to have a common userid and password image across these machines. TME 10 User Administration will be installed to manage groups, userids, and passwords. The SP User Management Services will be turned off.

An automounter will be used to provide access to all user home directories. TME 10 Software Distribution will be installed and used to distribute automounter map files. The SP Automounter Support and SP File Collection Services will be turned off.

Since the lab is mainly used as a test environment, applications are continuously installed and upgraded. TME 10 Software Distribution will be used to distribute application files to the appropriate workstations and SP nodes.

All systems will be monitored using the TME 10 Enterprise Console. The new PSSP T/EC Adapter will be installed on all SP nodes and control workstations to capture and forward events generated by the SP Event Manager to T/EC.

2.4.3 Deployment Planning

Planning to install Tivoli in this example environment, we decided on the following configuration:

- One TMR for the entire lab

  Since both RS/6000 SPs are small, we decided that one Tivoli Management Region for the entire lab would be sufficient. This TMR would include the RS/6000 SPs, as well as all of the standalone RS/6000 workstations. We do not plan to have a rapid growth rate of new nodes on the RS/6000 SPs or additional RS/6000 workstations, so we do not need to worry about outgrowing our TME server in the near future.

- TME server on RS/6000 workstation

  Since this is mainly an SP lab and the RS/6000 workstations are only minimally used during application testing, we will dedicate one of the RS/6000s to be our TME server. We will use this same server as our T/EC server.

  The current Local Token ring is not heavily used and can easily withstand the additional network traffic required by the Tivoli distributed database and management functions. We will access the SP nodes through the SP Ethernet adapters.

- TMR connections

  At this time, there is no advantage to connecting this TMR with other TMRs in the enterprise. The lab would like to maintain its autonomy. However, if regional procedures change such that a more global administrative policy is instituted, there may later be a need to connect this TMR with other TMRs in the region.

- Location of files

  The Tivoli library and binary files will be installed on each RS/6000 and on each SP control workstation. The SP nodes will NFS mount the Tivoli directories from their respective control workstation over the SP Ethernet connection. This will provide quick access to the data without compromising the local disk space on the SP nodes. As required by Tivoli, all database files will reside locally on each node.

- One top-level policy region for the TMR

  We will have one top-level policy region to govern the entire TMR. This will be used for lab-wide administration. For example, all user and group management will be done from this policy region.

- Policy sub-regions for each RS/6000 SP

  There will be one policy sub-region for each RS/6000 SP and one sub-region for all of the RS/6000 workstations. These policy sub-regions will contain the actual managed nodes representing the individual workstations and SP nodes. Although users have access to all machines in the lab, they typically limit their work to one of the RS/6000 SPs.

- Profile managers

The following profile managers will be created as our initial working set. Additional profile managers will be defined to support individual applications or groups of resources as the system administration evolves.

- One profile manager for each RS/6000 SP that contains as its subscribers all of the managed nodes representing the SP nodes of that system.
- One profile manager for each RS/6000 SP that contains two subscribers: the control workstation managed node and the profile manager containing the SP nodes.
- One profile manager that contains two subscribers: the profile managers containing each set of SP nodes. This profile manager will be used to distribute data common to all SP nodes or to perform common tasks on each node.
- One profile manager that contains all of the RS/6000 managed nodes as its subscribers.
- One profile manager that has three subscribers: the two profile managers representing the RS/6000 SPs and the profile manager with the RS/6000 workstations. This profile manager will be used to distribute common system and user management data.

The hierarchy of using profile managers as subscribers to other profile managers makes administration much easier when new managed nodes are added to the system. They need to only be added to the lowest level profile managers as managed endpoints.

Chapter 3. Installation

Before installing Tivoli products on your RS/6000 SP system, you should develop a TME installation plan to determine how the system will be managed within your Tivoli enterprise.

You will need to:

- Decide the location of TME servers
- Decide the distribution of SP nodes across TMRs
- Decide which Tivoli products you will be installing and whether they will be installed on selected managed nodes or on all nodes in the TMR
- Develop a topology of policy regions and profile managers so that your TME 10 Framework configuration reflects your administrative and organizational structure.

The importance of proper planning must be stressed in order to make your Tivoli installation more efficient and to avoid time-consuming and costly mistakes. See Chapter 2, “Planning and Design” on page 3 for planning information.

This chapter includes information on how to use your RS/6000 SP switch to do your initial Tivoli installation. The *TME 10 Framework Planning and Installation Guide* should be followed for detailed installation instructions. This chapter also includes information on maintaining Tivoli database consistency. Even though this information is not specific to the RS/6000 SP, it is included here to stress its importance in maintaining your Tivoli environment.

3.1 Installing Tivoli Using the SP Switch

This section contains instructions for installing a TME server on an RS/6000 SP node using the SP switch to transfer the installation image from a CD-ROM mounted on the control workstation.

Using the SP switch to install and manage the TME 10 Framework and other Tivoli applications is an alternative to using an Ethernet or other external network connection to distribute the data. You must consider the impact to your current production SP environment before using this approach. The switch is typically reserved for critical data and application communication where a high performance communication link is essential. However, if your production environment can tolerate the additional impact on the SP switch of the Tivoli communications, you will gain performance improvements by using the switch.

The following procedure was developed by Neerav Shah of IBM Global Services and was used in customer sites to install a TME server on an SP node transferring the installation image across the SP switch.

Use the following steps to install the TME server on an SP node over the high speed switch network:

1. Mount the TME 10 Framework CD on /cdrom on the control workstation:

   ```
   mount -rv cdrfs /dev/cd0 /cdrom
   ```

2. Export the NFS mount for /cdrom to the chosen TME server node (this can be done via the SMIT interface).

3. Mount the NFS filesystem on the TME server node:

   ```
   mount <CWS_hostname>:/cdrom /mnt
   ```

4. Edit the /etc/wlocalhost file to reflect the TME server's high speed switch hostname.

5. Export the DISPLAY variable on the TME server node to reflect the control workstation:

   ```
   export DISPLAY=<CWS_ip_address>:0.0
   ```

6. Enable the control workstation display. On the control workstation, enter:

   ```
   xhost +
   ```

7. Set the hostname and uname to be the high speed switch hostname on the TME server node:

   ```
   hostname <hps_hostname>
   uname -S <hps_hostname>
   ```

8. Create the following directory:

   ```
   /usr/local/Tivoli/install_dir
   ```

   and enter a cd command to change to this directory.

9. Ensure that there is enough disk space on the required filesystems:

   ```
   /usr/local/Tivoli  42860 KB
   /etc/Tivoli           36 KB
   /var/spool/Tivoli  11558 KB
   ```

   Note: Verify the current space requirements by reviewing the *TME 10 Framework Release Notes* packaged with your TME 10 software.

10. Run the preinstall script:

    ```
    /mnt/wpreinst.sh
    ```

11. Run the actual install script:

    ```
    ./wserver -c /mnt
    ```

12. From the TME 10 Desktop generated on the control workstation, make sure all the install options are correct (that is, the locations for all Tivoli libraries, binaries, and so on). Select the option to create directories on install.

13. Specify the following:

    - License key in the specified text gadget.
    - Encryption level to be NONE.
    - TMR policy region name (for example, SP).
    - The TME server hostname (high speed switch hostname).

14. Click on **Install & Close**.

15. When the install is complete, revert the hostname and uname of the TME server to their original values.

16. Edit the /etc/wlocalhost file and reflect the high speed switch hostname, if you have not already done so.

17. From the command line run the following command:

    ```
    odadmin environ get > /tmp/<any_filename>
    ```

    Edit /tmp/<any_filename> to reflect the following variable definition:

```
ALI_HOST_NAME=<high_speed_switch_hostname>
```

From the command line, run the following command:

```
odadmin environ set < /tmp/<any_filename>
```

18. Shutdown the TME 10 Desktop
19. Run:

    ```
    . /etc/Tivoli/setup_env.sh
    ```

20. Shutdown the oserv daemon:

    ```
    /etc/Tivoli/oserv.rc stop
    ```

21. Restart the oserv daemon:

    ```
    /etc/Tivoli/oserv.rc start
    ```

22. Start the TME 10 Desktop. From command line, type:

    ```
    tivoli
    ```

23. Perform backup of the Tivoli database. Refer to the *TME 10 Framework Planning and Installation Guide* for instructions on how to do this.

After completing these steps, you will have an SP node installed as your TME server. You can install the other SP nodes as managed nodes by following the normal installation procedures described in the *TME 10 Framework Planning and Installation Guide*. You must specify the switch host name of the SP node as the name of the node to be installed to force the installation across the switch. This will also cause all future TMR database communications, profile and data distributions, and so on, to occur across the switch.

If you decide at a later time that you do not wish to use the SP switch for your Tivoli communications, you can follow the directions for changing IP names and addresses described in the “TME 10 Maintenance and Troubleshooting” chapter of the *TME 10 Framework Planning and Installation Guide*.

3.2 Tivoli Object Database Consistency

The information in this section is not specific to executing Tivoli on the RS/6000 SP. However, the importance of maintaining Tivoli Object Database consistency and correctness is so critical to your Tivoli operation, that we felt it was necessary to emphasize the information here.

It is important to make regular backups of the Tivoli Object Database. The first reason for doing this is a common-sense one: the TME database contains important system management information that you want to protect from hardware or software failures.

The second reason is less obvious. When you install Tivoli products, the installation process is in fact performing a sequence of actions:

1. Installing code on the managed nodes
2. Executing local configuration programs
3. Updating configuration entries in the Tivoli Object Database

If there are any problems during installation, or if you need to reinstall a product, the easiest way to reverse the process is to restore the Tivoli Object Database from a backup prior to the installation.

3.2.1 Database Backup

There are several ways to backup your Tivoli Object Database. The following list describes some methods you can use:

Method 1 Use the CLI command wbkupdb directly. This command backs up and restores TME 10 databases:

```
wbkupdb -d /tmp/Tivoli_DB.bk sp2en0
```

where /tmp/Tivoli_DB.bk is the name of the backup file and sp2en0 is the name of the TME server.

Method 2 Issue the backup operation from the pull-down menu of the TME 10 Desktop. For details of this procedure, refer to the ITSO redbook *TME 10 Cookbook for AIX Systems Management and Networking Applications*.

Method 3 Use the Tivoli Task Library function to create a task to execute the CLI command wbkupdb, once or regularly. For details of this procedure, refer to the ITSO redbook *TME 10 Cookbook for AIX Systems Management and Networking Applications*.

Method 4 Use the cron utility to execute the CLI command wbkupdb regularly.

3.2.2 Database Restore

The following list describes two methods you can use to restore your Tivoli Object Database:

Method 1 Use the CLI command wbkupdb directly:

```
wbkupdb  -r -d /tmp/Tivoli_DB.bk sp2en0
```

where /tmp/Tivoli_DB.bk is the name of the backup file and sp2en0 is the name of the TME server.

Method 2 If you encounter a situation where you cannot use the wbkupdb command, or if the object dispatcher that is to be restored is not running (and presumably cannot be run because its database is corrupted or missing), or if you lost your root administrator, you can extract the database manually and put the files in the correct location in the database directory. The following is an example of how to rescue a TME server using the previously saved /tmp/Tivoli_DB.bk file for server sp2en0.

Note:

Before testing this example, make sure that the object dispatcher has been stopped.

```
. /etc/Tivoli/setup_env.sh
cd /worktmp/tivoli
tar xvf /tmp/Tivoli_DB.bk sp2en0
uncompress -c <sp2en0 | tar tvf -
cp imdb.bdb.restore $DBDIR/imdb.bdb
cp odb.bdb.restore $DBDIR/odb.bdb
cp odb.log.restore $DBDIR/odb.log
cp odlist.dat.restore $DBDIR/odlist.dat
```

3.2.3 Database Consistency Check

To check the consistency of the Tivoli Object Database, use the CLI command `wchkdb`. This command verifies and repairs problems in the TME 10 database. It does not affect system files; it only modifies resources in the TME 10 environment.

The first example checks and, if needed, repairs the TME 10 database. Object references are checked across TMR boundaries:

```
wchkdb -u -x
```

The second example checks object references in the current TMR only. No changes are made to the TME 10 database. Problems are, however, displayed to stdout and written to a binary output file, /tmp/check.out.

```
wchkdb -o /tmp/check.out
```

The third example reads the results from a previous run of `wchkdb` (/tmp/check.out) and updates the TME 10 database as needed:

```
wchkdb -u -f /tmp/check.out
```

3.2.4 Synchronizing Databases across TMRs

If you have a TME with multiple TMRs connected to each other, you must synchronize the Tivoli Object Database of each TME server after you have changed the configuration of TME, or at regular intervals.

The reason for doing this is that the name registry (Tivoli Object Database) is used as an intra-TMR name service, and when TMRs are connected, as an inter-TMR name service. To reduce the number of cross-TMR messages that must be sent during name lookups, the resource information of one TMR is maintained in the name registry of all connected TMRs.

During the initial connection process, the administrator is asked whether a resource update should be performed immediately upon connection. This is the only time that an update takes place automatically. To keep information on remote resources current in the local name registry, updates must be scheduled regularly.

TMR updates are always *pull* operations. This means that one TMR requests information from one or more connected TMRs, but it cannot push its current name registry to another connected TMR.

There are several ways to perform a TMR database update:

Method 1 Use the CLI command `wupdate`. The following example updates the local name registry with the *all* resource type from all other TMRs.

```
wupdate -r All All
```

Method 2 Issue the update operation from the pull-down menu of the TME 10 Desktop. For details of this procedure, refer to the ITSO redbook *TME 10 Cookbook for AIX Systems Management and Networking Applications*.

Method 3 Use the Tivoli Task Library function to create a task to execute the CLI command `wupdate`, once or regularly. For details of this procedure, refer to the ITSO redbook *TME 10 Cookbook for AIX Systems Management and Networking Applications*.

Method 4 Use the cron utility to execute the CLI command `wupdate` regularly.

Chapter 4. Event Management Integration

One of the most important issues of the integration of the RS/6000 SP and Tivoli is the event management integration. In PSSP 2.2, the Event Manager and the Problem Manager are the main components to realize specific RS/6000 SP event detection and handling.

Tivoli has two components that can be used for monitoring managed nodes: TME 10 Distributed Monitoring and TME 10 Enterprise Console. These components can only monitor the RS/6000 SP nodes as standard RS/6000 system resources. To monitor SP-specific resources, the following possibilities were investigated to forward events detected by Event Management and handled by Problem Management:

- The TME 10 Enterprise Console adapter program.

  The project team developed this program, tecad_pssp, at the PSSP lab in Poughkeepsie. It creates a direct link between the Event Manager and the TME 10 Enterprise Console using a new custom set of BAROC classes. The contents of this program are listed in Appendix D, “Source files for the PSSP T/EC Adapter” on page 153. Instructions to compile and create the executable module are listed there. Finally, a diskette is attached to this redbook, with a ready-to-use executable program, called tecad_pssp (the event adapter).

- The TME 10 Enterprise Console SNMP event adapter.

  Using the option in Problem Management to generate an SNMP trap, the trap is caught by the SNMP adapter and forwarded to the TME 10 Enterprise Console.

- The TME 10 Enterprise Console Logfile adapter.

  When an event is generated, a log file entry can be written to a log file. The Logfile adapter monitors this log file, and generates a Tivoli event when a new entry occurs.

- NetView/6000 for AIX as interface between RS/6000 SP and TME 10 Enterprise Console.

  Another way of using the SNMP mechanism is to use NetView/6000 for AIX (or another SNMP manager) as an intermediate station. The SNMP trap is sent to the NetView console and, using the NetView-Tivoli bridge, the NetView message is forwarded to a TME 10 Enterprise Console.

- The TME 10 Distributed Monitoring `wasync` command.

  The `wasync` command generates a string, which is sent to the TME 10 Distributed Monitoring console. The usage of this command is explained in Figure 35 on page 88.

- The TME 10 Distributed Monitoring SNMP monitoring tool.

  This option was not investigated because other SNMP interfaces investigated in this chapter offer more flexibility.

These options are pictured in Figure 2 on page 26.

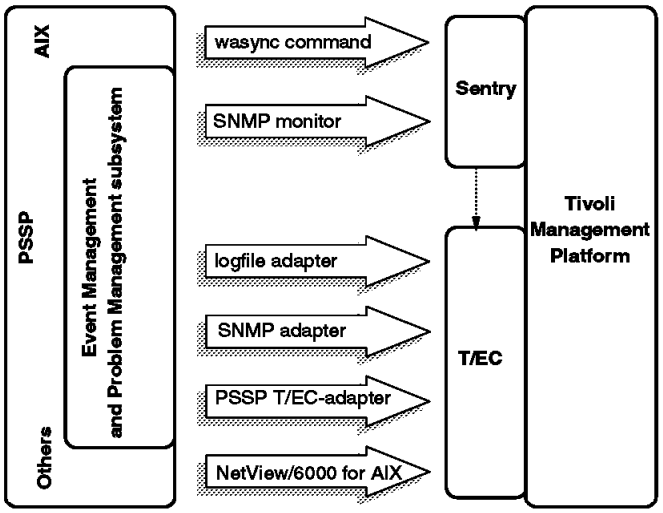


Figure 2. Available Integration Methods

4.1 Defining the TME 10 Enterprise Console

The TME 10 Enterprise Console, known as T/EC, is one of Tivoli′s availability management applications. As a management tool, it assists in maintaining high availability of the myriad of networks, system applications, and databases found within the scope of an enterprise, and provides a centralized point of control for all critical messages stemming from these resources.

The computing resources in the distributed environment generate messages in a variety of formats. The function that captures these messages is called an *event adapter*. Each event adapter installed on or near the monitored computing resources is familiar with the structure of that raw data and transforms the information it receives by parsing and restructuring it before sending it to the T/EC event server in a T/EC-acceptable format, a special structured format named BAROC (Basic Representation of Object in C). BAROC is a simple event class structure that allows us to separate the important elements of a message into a number of pieces of information called slots. Each event is defined as a member of a class.

A number of different types of T/EC event adapters may exist on any TCP/IP-connected system, not only on TME managed nodes. Besides generating the events and sending them to the T/EC event server, the adapters are also responsible for filtering out extraneous information so as to forward only significant events to the T/EC event server. This helps reduce the demand on

the network and the amount of processing done by the T/EC event server when the computing resources are sending out large numbers of messages. For details on the procedures to install and use T/EC, refer to the ITSO redbook, *TME 10 Cookbook for AIX Systems Management and Networking Applications*.

Table 2 shows an overview of the different methods we investigated.

Table 2. Event Integration Methods. This table summarizes the event integration methods explored in this book.

Method	Producer[1]	Consumer[2]	Transport	Advantage (+) Disadvantage (-)
PSSP T/EC Adapter	Invoke tecad_pssp command from PSSP Problem Management	T/EC	Tivoli framework (secure) or sockets (insecure)	(+)Easy to use (+)Flexible parsing in T/EC ()Support available through PRPQ
SNMP Adapter	SNMP trap from PSSP Problem Management or AIX error log	T/EC SNMP Adapter	SNMP trap -> T/EC event	(+)Trapids for severities or other user-defined information (+)Traps can be used for both AIX (errorlog) and PSSP events (-)Event info is all strings (-)Difficult to parse/filter (-)Difficult as target
Logfile Adapter	T/EC Logfile Adapter scans system log file (syslog)	T/EC	Tivoli Framework (secure) or sockets (insecure)	(+)Any file can be used as source (+)GUI to configure (-)Log file format must be known
NetView/6000 for AIX	SNMP trap from PSSP Problem Management or AIX error log	NetView forwards to T/EC	SNMP trap -> T/EC event	(+)May be good choice if using NetView (+)NetView can be used to filter (-)All data is strings
TME 10 Distributed Monitoring wasync command	Invoke wasync command from PSSP Problem Management	TME 10 Distributed Monitoring	TME 10 Distributed Monitoring	(+)Cheap and simple (+)Can be forwarded to T/EC (-)Inflexible

4.2 The PSSP T/EC Adapter

The PSSP T/EC Adapter is a tool that forwards events generated by the PSSP Event Manager subsystem to TME 10 Enterprise Console. The PSSP T/EC Adapter receives the PSSP events, extracts all the event information, and formats it into a T/EC event notification automatically. This tool greatly simplifies the forwarding of events, since it does not require any other program or script to parse the event strings. The PSSP T/EC Adapter receives all the event information from the Event Manager subsystem and formats it into T/EC events using the PSSP_EVENT classes, which are defined in the pssp_classes.baroc file. Using the `tecad_pssp` program allows the system administrator to subscribe for events using the Event Perspective GUI or the `pmandef` command, and have this event forwarded to T/EC, thus enabling event forwarding in an integrated GUI environment.

4.2.1 How the PSSP T/EC Adapter Forwards Events

The PSSP T/EC Adapter (`tecad_pssp`) forwards events created in PSSP by parsing all the environment variables that are provided by the Problem Management subsystem, and formatting them into T/EC event classes. It can forward an event to a T/EC server that is in or outside the SP system, and it is recommended that the adapter be installed on the control workstation.

[1] Application that produces event.

[2] Application where event is sent and processes event.

The recommendation to install the tecad_pssp command only in the control workstation is made for the following two reasons:

1. The tecad_pssp command utilizes a configuration file (tecad_pssp.cfg), which contains information about the T/EC server to which the event is to be forwarded. By restricting the installation to the control workstation we ensure the consistency of the communication settings throughout the SP system.
2. The tecad_pssp command can utilize a *secure* transport to forward the event to the T/EC server (utilizing TME communications). If this mode of communication is desired, the node where tecad_pssp runs must be configured as a Tivoli Managed Node. A savings in disk space and a gain in performance can be achieved by restricting the forwarding of events to the control workstation, requiring only one node in the SP system to be configured as a managed node.

The event subscription can be done using the PSSP Event Perspective or the PSSP Problem Management Subsystem. The event forwarding process is illustrated in Figure 3.

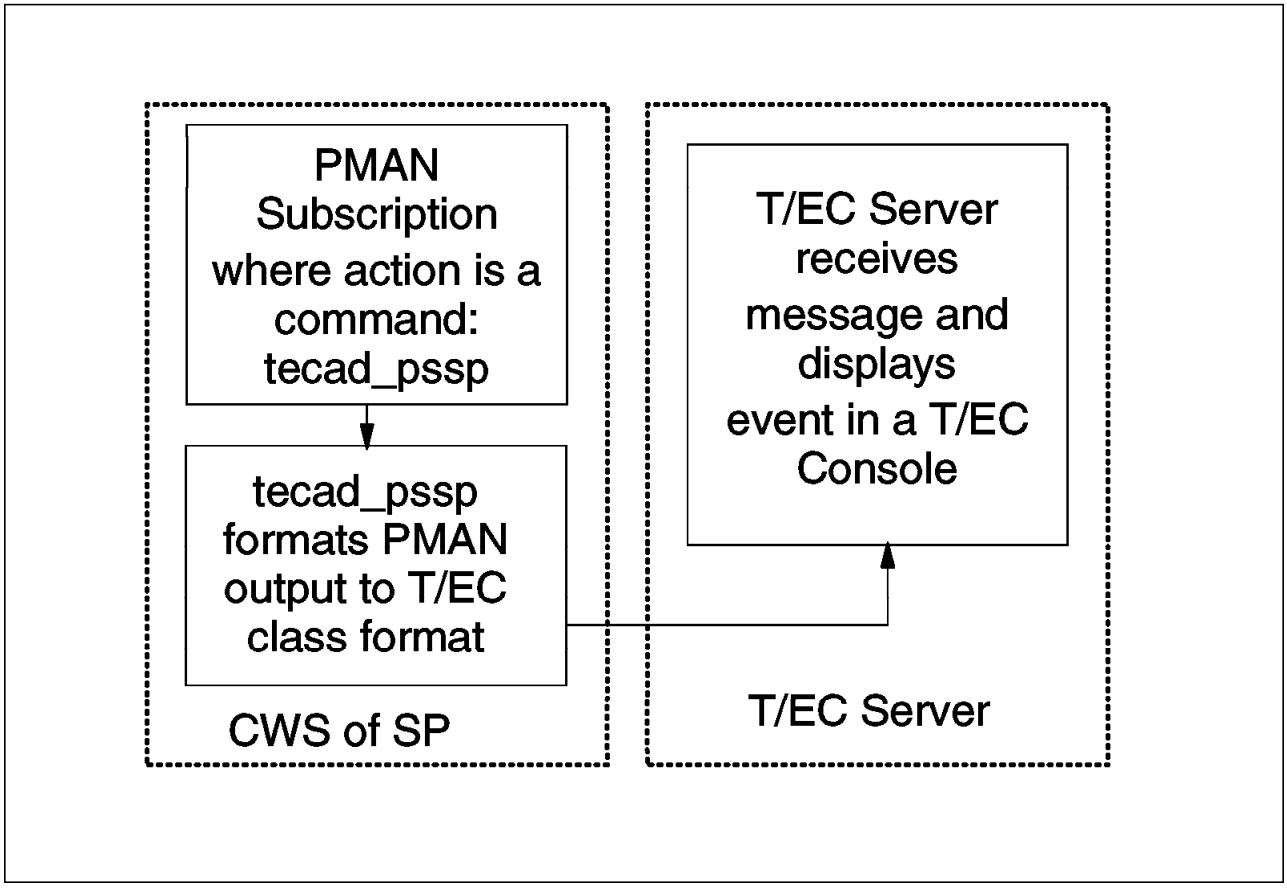


Figure 3. PSSP T/EC Adapter Control Flow

4.2.2 Using the PSSP T/EC Adapter

The PSSP T/EC Adapter was built using the standard Tivoli Event Integration Facility (EIF). The procedure for forwarding PSSP events to the TME 10 Enterprise Console is as follows:

1. Install the pssp_classes.baroc file in the T/EC server.
2. Make an event source, group and filter for the PSSP events in the T/EC server.
3. Install the `tecad_pssp` program in the control workstation of the SP system.
4. Make an event subscription using the PSSP Event Perspective GUI.

The last step is repeated for every event that is to be forwarded to the Tivoli Enterprise Console.

4.2.3 Event Classes Defined for PSSP Events

The event classes defined in the pssp_classes.baroc file accommodate all the standard PSSP Event Manager events into 20 classes. The primary objective in the design of these classes was to group the PSSP Event Manager Events that share common attributes, so as to produce a minimal set of classes that could accommodate all events. It is quite possible to extend these classes to accommodate user-defined events and user-defined resource variables. A user interested in extending these classes should refer to 4.2.10, “Extending the PSSP Event Classes” on page 35. The rvclasses.cfg file also contains comments that show how the Event Management resource variables were mapped into T/EC classes. The following illustration shows the class hierarchy defined for the PSSP Events:

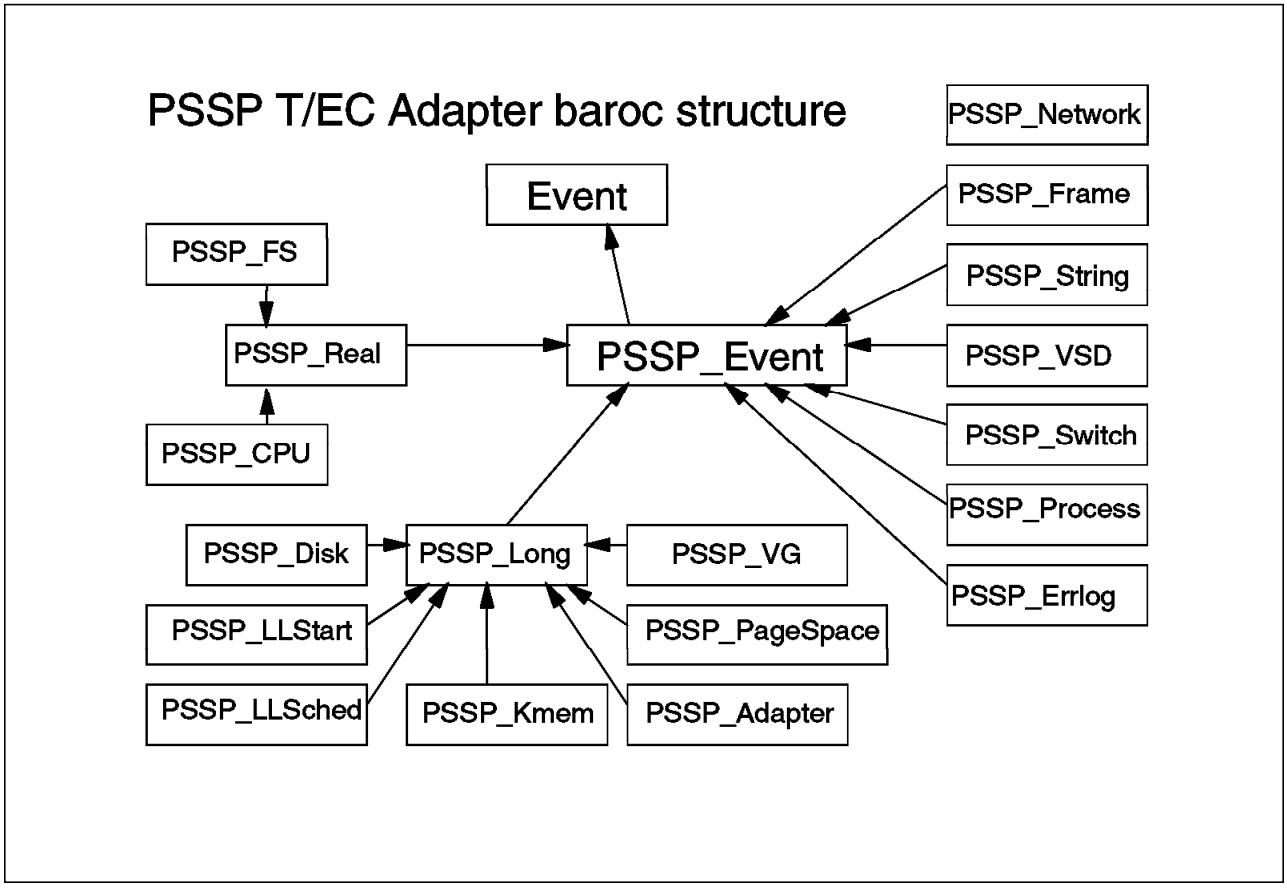


Figure 4. PSSP T/EC Adapter Class Hierarchy

4.2.4 Installing the PSSP T/EC Adapter Classes in the T/EC Server

The first step for receiving PSSP events in the T/EC console is to install the classes defined in the tecad_pssp.baroc file into a rule base. This procedure is similar to the one followed in 4.3, “Using the TME 10 T/EC SNMP Adapter” on page 45, and is covered in detail in *TME 10 Cookbook for AIX Systems Management and Networking Applications*. The generic procedure to configure the T/EC server is:

1. Create a rule base in Event Server Console:

   ```
   wcrtrb -d /your_rulebase_path pssp
   ```

   Also refer to the `wlsrb` command to find the directory of existing rule bases.

2. Copy the default rule base (class definition and rule) into the new rule base:

   ```
   wcprb Default pssp
   ```

3. Import a new class file (*.baroc.) into the rule base:

   ```
   wimprbclass pssp_classes.baroc pssp
   ```

4. Compile and load the new rule base:

   ```
   wcomprules pssp
   wloadrb pssp
   ```

5. Shut down and restart the event server:

   ```
   wstopesvr
   wstartesvr
   ```

4.2.5 Configuring the T/EC Server to Receive PSSP Events

In this step, create an event source named “PSSP” and create an event group that you will use to monitor SP events.

Then add an event filter to your event group. Utilize the superclass PSSP_EVENT as the filtering class for all PSSP events. You will then receive all the events generated by the SP system with this event filter.

If you would like to process only a subset of the PSSP events, it is recommended that you utilize rules that match the resource_variable slot to the EM resource variables which you are interested in. You cannot filter the SP events utilizing the PSSP class names, because they are grouped by type (for example, integer, real, string), and not by event nature (for example, network, file system, process death, and so on). The grouping of the PSSP events according to event nature is done in the resource variable name space, in order to be consistent with the Event Manager event hierarchy. Consult the *IBM PSSP for AIX Event Management Programming Guide and Reference* for information on the resource variable hierarchy. A listing of the PSSP 2.2 resource variables is included in Appendix A, “Event Management Resource Variables” on page 139.

An alternative way of filtering the events in the T/EC server is by applying rules to the administrator slot or the subscription_handle slots, and filtering for the subscription you made in the SP system.

In order to configure the T/EC server to receive PSSP events, you must:

1. Create a new event source of type “PSSP.”
2. Create a new event group that you will use to monitor SP events.
3. Add a new event filter to this event group, selecting the PSSP_EVENT class as the class to filter, and “PSSP” as the event source.
4. Assign the new group to your event console.

You can test if the classes for PSSP events were correctly loaded, and the source and event filters were properly defined, by using the wpostemsg command:

```
wpostemsg -m "test" PSSP_EVENT PSSP
```

This should create an event in your T/EC console.

4.2.6 Compiling the PSSP T/EC Adapter in the RS/6000 SP

The PSSP T/EC Adapter can reside anywhere in the SP system, but it is recommended that it be installed at the control workstation, since the control workstation has access to all partitions in the system. The distribution disk attached to this book contains the source code for the PSSP T/EC Adapter, which is written in C (tecad_pssp.c), as well as a precompiled executable version of the adapter.

It is only necessary to compile this code and link it with the appropriate EIF libraries if you are using a different version of TME 10 Enterprise Console, or if you have made changes to the original source. The libraries that you use depend on which communication model you want the adapter to use. For *unsecure* communications, for example, communications that are not transmitted using the T/EC channel, you will need to have the EIF libraries installed. Compile the source code with the following command:

```
gcc -g tecad_pssp.c -o tecad_pssp
    -I/usr/include/local/Tivoli/include
    -L/usr/local/Tivoli/lib/aix4-r1
    -lteceif -lg++ -lm
```

To compile the PSSP T/EC Adapter code for *secure* communications (for example, using the T/EC channels for transmitting the events), your control workstation needs to be installed with T/EC at least as a managed node. You will need to have the ADE libraries installed. Compile the source code with the following command:

```
gcc -g tec_adapter.c -o tecad_pssp
    -I/usr/local/Tivoli/include/aix4-r1
    -L/usr/local/Tivoli/lib/aix4-r1
    -ltec -las -las_imp -ltds -lui -ltas -ldes
    -ltmfimp -ltmf -lthreads -lg++ -lm
```

Note:

The command used to compile the tecad_pssp code was gcc, the GNU C-Compiler. The gcc command is not available with AIX, but can be downloaded from the internet. A suitable site to refer to for getting the GNU C-Compiler is: http://www.fsf.org, the home page of the Free Software Foundation.

The PSSP T/EC Adapter package contains a script for compiling the tecad_pssp program, called makeit. You can build a secure or unsecure tecad_pssp by typing:

```
makeit secure
```

or:

```
makeit unsecure
```

If you don't specify the channel option, unsecure is assumed.

4.2.7 Installing the PSSP T/EC Adapter in the RS/6000 SP

Once you have an executable version of the tecad_pssp command, you should move the command and the tecad_pssp.cfg file to some directory in the control workstation. If the control workstation is a managed node that has other adapters, you should put both files in the /etc/Tivoli/tecad/bin directory. If not, we recommend putting the tecad_pssp command in /usr/lpp/ssp/bin, and the tecad_pssp.cfg file in /usr/lpp/ssp/config.

The next step in the installation process is to run the install_agent command. This puts the mapping of the Event Management resource variables to T/EC classes in the System Data Repository (SDR). The usage of install_agent is:

```
install_agent config_filename
```

where config_filename is a configuration file that contains the mapping. The configuration file rvclasses.cfg is part of the PSSP T/EC Adapter package and is available on the diskette attached to this redbook. You must first run the install_agent command with this file:

```
install_agent rvclasses.cfg
```

You can also use the install_agent command to extend the classes defined by rvclasses.cfg with your own configuration file. See 4.2.10, "Extending the PSSP Event Classes" on page 35 for details on how to do this.

4.2.8 Using the tecad_pssp Command

The tecad_pssp command was designed to be executed by the PSSP Problem Management subsystem. It should not be executed by any other subsystem, since it depends on environment variables that are exported by the Problem Management daemon, pmand. Therefore, to forward PSSP events using the tecad_pssp command, you need to make a Problem Management subscription using either the PSSP Event Perspective GUI (the procedure is outlined in 4.2.9.1, “Making Subscriptions Using Event Perspectives” on page 34), or using Problem Management directly (see 4.2.9.2, “Making subscriptions with the pmandef command” on page 35). In either case, you should select tecad_pssp as the command to run for that subscription, and provide the appropriate parameters.

The tecad_pssp syntax is:

```
tecad_pssp [-l path/filename]
           [-Cc]
           [-m text]
           [-a tiv_admin_name]
           [-s severity]
           [-p port]
```

The flags of the command have the following meaning:

-l is the path/filename of the configuration file. The default value is /usr/lpp/ssp/config/tecad_pssp.cfg. The only required value in this file is the *ServerLocation* parameter. It should be one of the following:

- ServerLocation=hostname.domain, for secure communications
- ServerLocation=@ServerName, for a managed node over a TME channel
- ServerLocation=@ServerName#RegionName, for secure transport in connected TMRs

You should consult *TME 10 EIF User's Guide* for the correct values of the other configuration parameters.

-C for connection-oriented protocol

-c for connectionless protocol (default)

-m adds *text* to the message field of the event

-a adds *admin* in the T/EC_administrator field of the event

-s sets the severity of the event to *severity*. The legal values for the severity are the following strings:

FATAL

CRITICAL

WARNING

MINOR

HARMLESS

INDETERMINATE

If an illegal value is used, the default UNKNOWN is utilized.

-p sets the communication port number to *port*. Note that you can also set the port number in the configuration file.

Once tecad_pssp is compiled and the install_agent command has run without errors, the tecad_pssp command is ready to be tested. The following illustration shows a shell script that can be used for that purpose:

```
export PMAN_IVECTOR=ProgName='marcos_test;UserName=root;NodeNum=0'
export PMAN_LOCATION=0
export PMAN_PRED=X@0==0
export PMAN_PRINCIPAL=root.admin@PPD.POK.IBM.COM
export PMAN_RVCOUNT=3
export PMAN_RVFIELD0="CurPIDCount=0"
export PMAN_RVFIELD1="PrevPIDCount=1"
export PMAN_RVFIELD2="CurPIDList=123,456"
export PMAN_RVNAME=IBM.PSSP.prog.pcount
export PMAN_RVTYPE=sbs
export PMAN_TIME='Thu May 29 12:08:17 1997'
tecad_pssp -m "This is a test" -a myadmin -s WARNING
```

Figure 5. The test script test_agent

The test_agent script should make the tecad_pssp command forward the event to the T/EC server defined in the tecad_pssp.cfg file. If you have an Event Console defined to receive PSSP events (see also 4.2.5, “Configuring the T/EC Server to Receive PSSP Events” on page 31) you should receive the test event. If you do not receive it, you may have incorrectly defined the event filter or event group. You can check if the event was received at the T/EC console by using the wtdumprl command.

4.2.9 Making Event Subscriptions in the RS/6000 SP

The High Availability Infrastructure allows you to define events in two different ways:

1. By using the Event Management Perspectives (spevent)
2. By using the *pmandef* command, part of the Problem Management subsystem

4.2.9.1 Making Subscriptions Using Event Perspectives

Using Event Management Perspectives makes the administration and management of the events in the SP system relatively easy.

Detailed instructions for event subscription and management can be found in *IBM PSSP for AIX Administration Guide*. Once the Event Perspectives is started, the way to create an event is to select: **Actions→Event Definitions→Create**. This will bring up the event definition window illustrated in Figure 6 on page 39, where you can select the parameters for the event definition. Once the definition is created, the response options to the event definition should be defined. The illustration shown in Figure 8 on page 41 shows an example of the Response Options screen in Event Perspectives. You should now select **Take Actions when event occurs**, and enter the tecad_pssp command in the command window, as illustrated in this example. Do not forget to give a full path to tecad_pssp and to supply a full path to your configuration file, if you are not using the default. Refer to 4.2.11, “An Example: Using the PSSP T/EC Adapter” on page 37 for a complete example of the subscription process. Refer to 4.2.12, “Debugging the

Event Generation and Reception" on page 45 for information on debugging subscription problems.

You should repeat the same thing for the rearm command if you wish. On the lower part, you should select the control workstation (Node 0) as the node where to run the command. Again, you could install the tecad_pssp command on all nodes, but our recommendation is to utilize the control workstation as a central point for event forwarding, since it is accessible by all partitions.

4.2.9.2 Making subscriptions with the pmandef command

The second way to create event subscriptions is by directly using the PSSP pmandef command. Consult the pmandef command description in *IBM PSSP for AIX Administration Guide*. A general example could be as follows:

```
pmandef -s example1
        -e "AnyResourceVariable;Any InstanceVector;AnyPredicate"
        -c "$AGENT_PATH/tecad_pssp -l $CONF_PATH/tecad_pssp.cfg"
        -r "AnyRearmPredicate"
        -C "$AGENT_PATH/tecad_pssp -l $CONF_PATH/tecad_pssp.cfg"
        -n 0
```

Note

It is recommended to run the event response command from the control workstation. This saves installation efforts and keeps the management of your environment easier. This results in the flag -n 0, indicating that the command needs to be run on Node 0, the control workstation.

4.2.10 Extending the PSSP Event Classes

If you want to extend the tecad_pssp command to handle user-defined events, do the following:

1. Create the new resource variable and make it known to the Event Management subsystem (see details in the *IBM PSSP for AIX Event Management Programming Guide and Reference*).
2. Map this new resource variable into some T/EC event class You can use one of the T/EC classes created in the pssp_classes.baroc file, or create a new one. If you create a new T/EC class, you will need to load it in the T/EC server.
3. Make the new class known to the tecad_pssp command, if you decide to create one. To do this:
   a. Create a new configuration file with a line that maps your new resource variable to the new class name and the new class number. The class numbers are defined by the enum statement in the tecad_pssp.c file, (the numbers 0-17 are already taken).
   b. You need to make the new class number known to the tecad_pssp command. You should include it in enum class_names_t. You can use the = operator to define your class number. For instance, if you are going to define the class name PSSP_YOURCLASS, and you want the class number to be 20, add the following to enum class_names_t:

```
typedef enum
{
  PSSP_LONG = 0,
  PSSP_REAL,
  PSSP_ADAPTER,
  PSSP_PROCESS,
  PSSP_STRING,
  PSSP_NETWORK,
  PSSP_FRAME,
  PSSP_SWITCH,
  PSSP_VSD,
  PSSP_DISK,
  PSSP_FS,
  PSSP_KMEM,
  PSSP_VG,
  PSSP_CPU,
  PSSP_PAGESPACE,
  PSSP_ERRLOG,
  PSSP_LLSTART,
  PSSP_LLSCHED = 17,
  PSSP_YOURCLASS = 20
} class_names_t;
```

Also, add a line in your new configuration file defining the new resource variable to class name/number mapping. For instance, if your resource variable is named:

```
Your.resource.name
```

Note:

Refer to *IBM PSSP for AIX Event Management Programming Guide and Reference* for the resource variable naming convention.

Then you create a configuration file with the following line:

```
Your.resource.name          PSSP_YOURCLASS  20
```

Note:

You may not need to define a new class. Look at the classes already defined in the pssp_classes.baroc file for one that matches the slots needed for your resource variable. For example, if your resource variable only needs a node number and returns an integer value, you could use the PSSP_LONG class. You should only define new classes if your resource variable utilizes unique parameters or return values.

4. If you do not define a new class for your resource variable mapping, you only need to change the rvclasses.cfg file. For example, if your resource variable utilizes the PSSP_LONG class, you create a configuration file with the following line:

   ```
   Your.resource.name      PSSP_LONG       0
   ```

   Note that you need to know the number of the class, which is defined in the enum class_names_t. Since the PSSP_LONG is the first one, it is 0, PSSP_REAL is 1 and so on.

5. After you create the new configuration file, you need to run `install_agent` again:

```
install_agent your_configuration_file_name
```

To ensure that your class mapping is added to the SDR, you can issue the following command:

```
SDRGetObjects tecad_pssp_Class rv_name=Your.resouce.name
```

And the command should return a mapping like the one previously described.

6. You should now be able to create an event subscription for your new resource variable, select `tecad_pssp` as the command to run, and receive a notification on the T/EC console of the proper class type.

4.2.11 An Example: Using the PSSP T/EC Adapter

This section demonstrates how the integration of the event management capabilities of the PSSP 2.2 Problem Management subsystem and the TME 10 Enterprise Console can be done in an integrated GUI environment using the PSSP T/EC Adapter.

The first step is to subscribe to the event of interest using the Event Perspective GUI. In this particular example, we will utilize the process liveness event, which monitors the presence of particular processes.

Using the Event Management process monitor facility of PSSP 2.2, we can define events related to the “death” (when the process exits or is killed) and “birth” of processes (when the process is executed). The Event Manager resource variable associated with the process monitor is named IBM.PSSP.Prog.pcount.

In our example, we will be monitoring the birth and death of a process named “testing.” These conditions can be observed by monitoring the number of processes currently running with the name “testing,” which were started by a certain user (in our example, we will monitor processes associated with the root userid), on a specific location (node). These parameters are specified in the instance vector for the IBM.PSSP.Prog.pcount variable.

We defined two events associated with the “testing” process, the arm event and the rearm event, as follows:

1. The arm event will be triggered by the death of the “testing” process; that is, when the total count of processes with the name “testing” under user root on Node 3 is zero. The return value of the IBM.PSSP.Prog.pcount resource variable is a structured byte string (see the Event Manager manual for the definition of structured byte strings), with the following fields:
   - Current count: the number of processes currently running
   - Previous count: the number of processes running in the last observation interval
   - Current list: the list of PIDs (process IDs) of the processes currently running

   We will define the arm event to trigger when there are no copies of the “testing” process running. Since the current count is the first field in the structured byte string, the predicate for this is:

   ```
   X@0==0
   ```

   (that is, field 0 equates to 0).

2. The rearm event will be triggered when we have exactly one instance of the "testing" process running. The predicate for this is:

   ```
   X@0==1
   ```

With these definitions, one arm event will be issued every time that there are no instances of "testing" running, and one rearm event will be issued every time exactly one copy of "testing" is started. This example is very useful for monitoring critical processes in your system. By making this kind of subscription and receiving such events in your TE/C console, you could then define rules and tasks which could re-start the critical processes.

The steps for making the subscription just described are as follows:

Step 1 Bring up the Event Perspective GUI. Then select the lower panel, **Event Definitions**, to enable the proper menus for event creation. Then select **Actions**→**Event Definitons**→**Create**. You will get a blank event definition window.

Step 2 Fill the event definition as illustrated in Figure 6 on page 39.

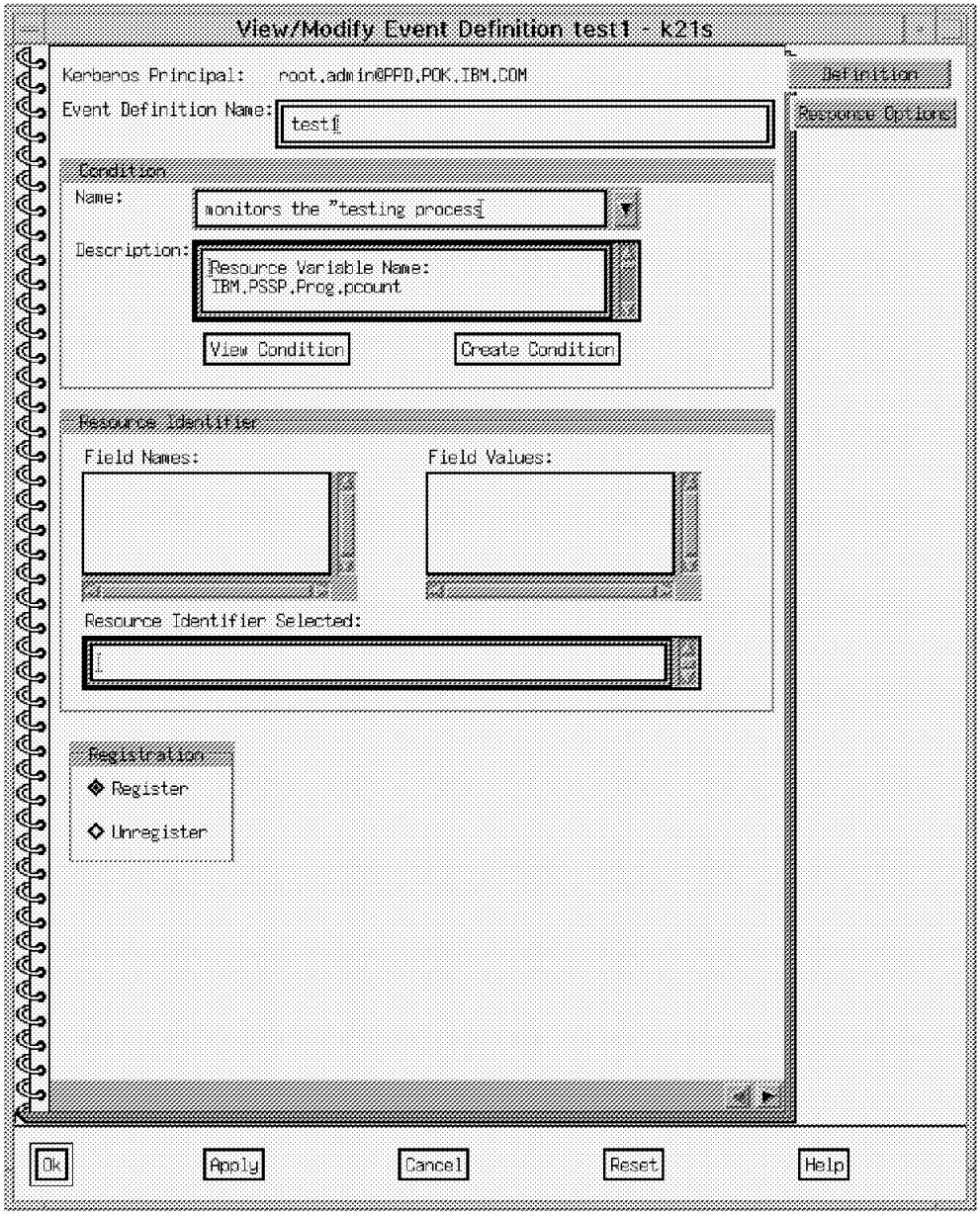


Figure 6. Event Perspective Event Definitions Window

Step 3 Now click on the **Create Condition** button, fill in the condition page as illustrated in Figure 7 on page 40, then click **Ok**.

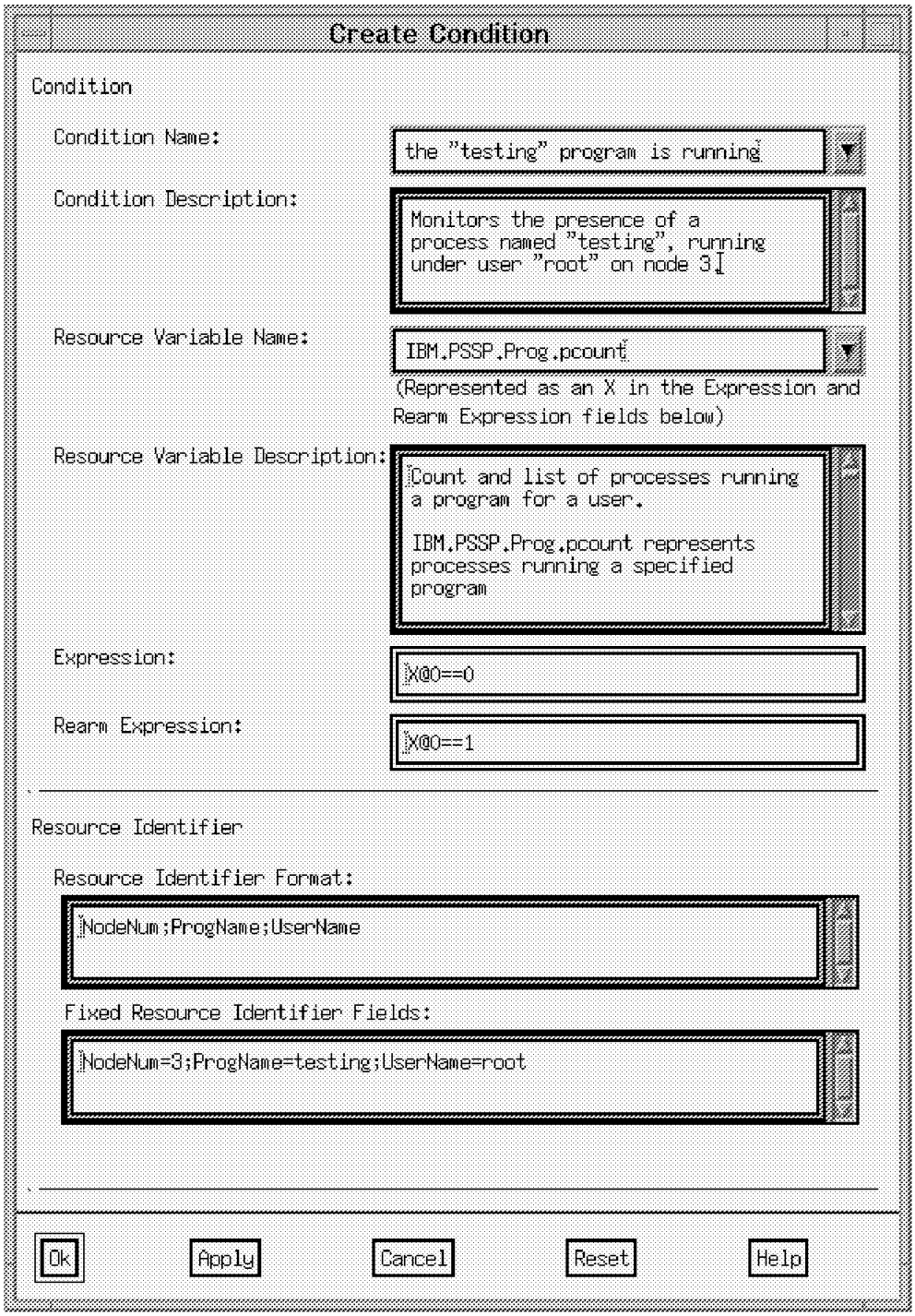


Figure 7. Event Perspective Create Condition Window

Step 4 After you click **Ok** in the Condition Page, you get back to the Event Definition page. You should then select the **Response Options** marker in the note book, and fill the reponse option as shown in Figure 8 on page 41.

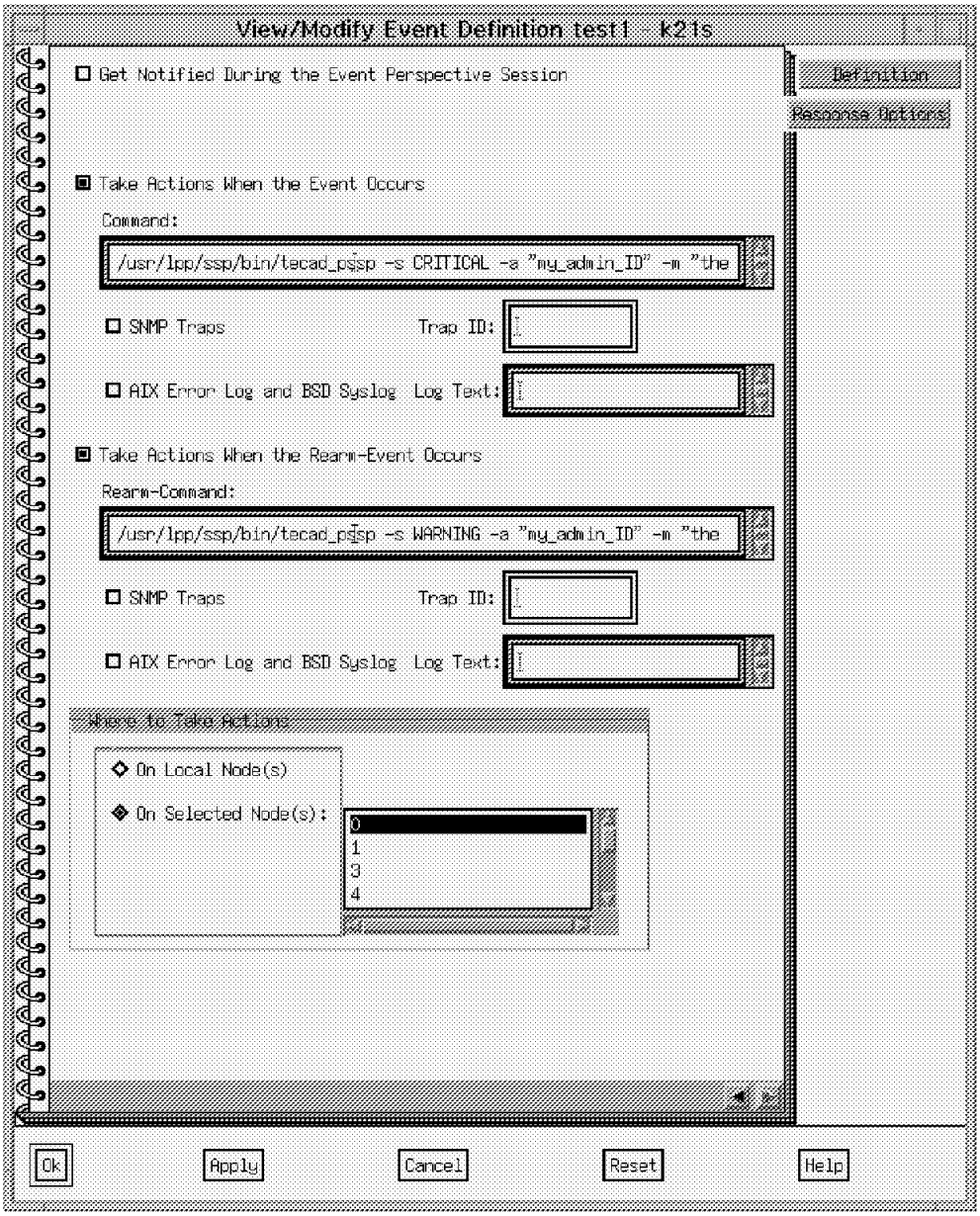


Figure 8. Event Perspective Response Options Window

Step 5 Now click on the **Create** button, and the subscription will be made to the Event Management System.

You are now ready to test your subscription. Note that in our example we are monitoring node number 3, and running the tecad_pspp, as a result of the condition, on Node 0 (the control workstation). To test the subscription you will have to write a little program which basically just hangs on a loop, and call it “testing.”

Step 6 Test the event generation by starting the “testing” program with the same userid which you have subscribed for, in the node which you subscribed for. (In our case, we subscribed for user root running the “testing” process on node 3. You can change the subscription to fit your environment.)

To test the subscription, you run the “testing” command for a couple seconds, and then kill it. This should create an arm and a rearm event.

Important: You should verify that the event was really generated and properly subscribed for. One effective way to make this check is to go to the control workstation and type the command:

```
#lssrc -ls pman.your_partition_name
```

where your_partition_name is the partition name for the node for which you subscribed. In our case, this command shows the following information:

```
[k21s][/usr/lpp/ssp/bin]> lssrc -ls pman.k21s

Subsystem         Group            PID     Status
 pman.k21s        pman             21720   active

 pmand started at: Wed May 28 19:13:12 1997
 pmand last refreshed at:
 Tracing is off
 ============================================================
 Events for which registrations are as yet unacknowledged:
 ============================================================
 ============================================================
 Events for which actions are currently being taken:
 ============================================================
 ============================================================
 Events currently ready to be acted on by this daemon:
 ============================================================
 ----------------- test1 ----------------
 Currently ACTIVE
 Client root root.admin@PPD.POK.IBM.COM at k21s.ppd.pok.ibm.com
 Resource Variable: IBM.PSSP.Prog.pcount
 Instance: NodeNum=3;ProgName=testing;UserName=root
 Predicate: X@0==0
 Command to run: /usr/lpp/ssp/bin/tecad_pssp -s CRITICAL -a "my_a
 dmin_ID" -m "the testing process died" > /tmp/tecad.log
 Has run 1 times
 Last status: normal termination, exit status = 0
 Last run at: Wed Jun  4 14:53:28 1997
 Rearm predicate: X@0==1
 Command to run on rearm event: /usr/lpp/ssp/bin/tecad_pssp -s WA
 RNING -a "my_admin_ID" -m "the testing process is back" > /tmp/tecad.log
 Has run 1 times
 Last status: normal termination, exit status = 0
 Last run at: Wed Jun  4 15:36:07 1997
```

As you can see, the output of this command contains all the information about our subscription, and also tells how many times each arm/rearm event was triggered. It is essential that you verify that the event is being properly triggered at this point. If it is not, you must check your subscription. After you ensure that the event is being triggered, you can launch your T/EC console and verify that the events are being received.

Step 7 If you have not yet done so, define the PSSP event source and an event group which utilizes the PSSP_EVENT class as a filter. Then assign this event group to your Event Console.

Step 8 Repeat the event generation procedure in **Step 5**. Use the `lssrc` command to verify that the `tecad_pssp` command was run.

Step 9 You should now receive events from the PSSP source. In our case, the arm and rearm events will appear as shown in Figure 9 on page 43 and Figure 10 on page 44.

Figure 9. Arm Event Received from the PSSP Source

Figure 10. Rearm Event Received from the PSSP Source

4.2.12 Debugging the Event Generation and Reception

If you fail to receive events, check the following:

- Check your event subscription and test the event generation by forcing the event. Then use the `lssrc` command (as in **Step 6** in 4.2.11, "An Example: Using the PSSP T/EC Adapter" on page 37) to verify that the `tecad_pssp` command is being run.
- Use the `wtdumprl` command in the T/EC side to see if you are getting any event notifications from the PSSP source. If you are, the problem is not in the SP system. You should then check your event source, event group, and event filter definitions, as well as the event group assignments in your console. If you are not getting anything, the problem may be in the SP system side.
- If you suspect that the `tecad_pssp` command is being run, but nothing is being generated at the T/EC side, check if you have the proper configuration file installed (tecad_pssp.cfg), and that it points to your T/EC server. Use the `test_agent` command to force the execution of the `tecad_pssp` command (see Figure 5 on page 34). Also, you may want to check for network connectivity.

4.3 Using the TME 10 T/EC SNMP Adapter

PSSP 2.2 includes the Problem Management subsystem that provides an infrastructure for acting on problem events within the SP system. The Problem Management subsystem lets you subscribe to Event Management events, and to specify an SNMP trap as a response to an event.

The ability to issue an SNMP trap in response to an event allows you to report problem events occurring in your SP system directly to TME 10 Enterprise Console, or indirectly to an existing SNMP network manager such as NetView/6000 for AIX.

There are several ways to forward SNMP events from AIX to the TME 10 Enterprise Console. In Figure 11 on page 46, an overview is displayed of the possible ways to forward SNMP traps to the TME 10 Enterprise Console.

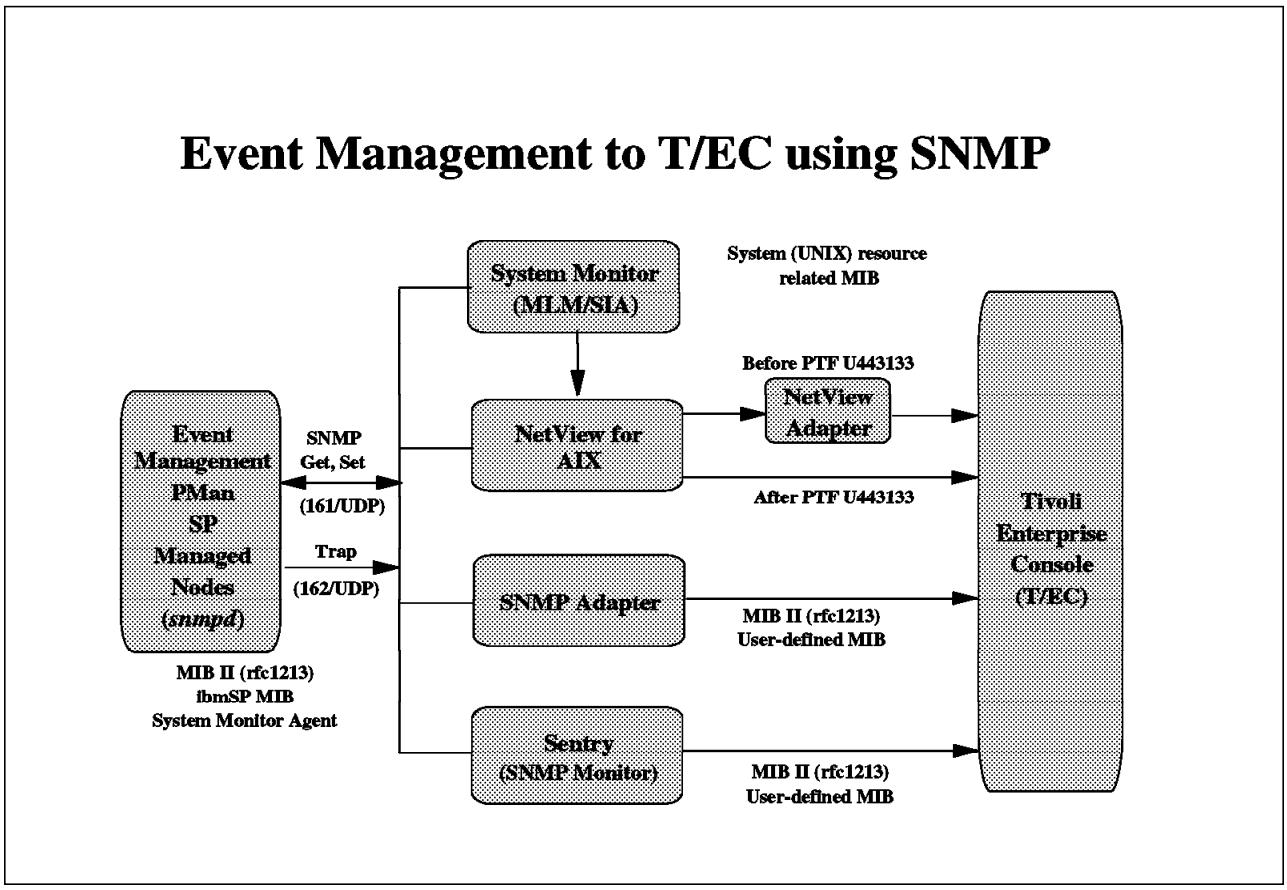


Figure 11. SNMP Connections

In this chapter, the connection to T/EC with TME 10 T/EC SNMP Adapter is explained. For an explanation of the connection using NetView/6000 for AIX, refer to 4.5, “Using NetView/6000 for AIX and TME 10 Enterprise Console” on page 75.

The TME 10 Distributed Monitoring SNMP Monitor connection and the System Monitor (MLM/SIA) were not investigated during the writing of this redbook.

4.3.1 Adapter Installation

The Tivoli documentation recommends that, by default, event adapters should be installed on the host that generates the SNMP traps.

In our situation, this is Event Perspectives on the control workstation. Therefore, the adapter was installed on the control workstation to receive SNMP traps generated by Event Perspectives. The adapter software is distributed with T/EC and can be installed following the directions in the *TME 10 Enterprise Console Event Adapter Guide: SNMP*. The contents of this guide should be read and understood before proceeding with this section.

Since the SNMP trap adapter listens on UDP socket 162 for incoming SNMP traps, you should check that the following entries exist in the /etc/services file:

```
snmp            161/tcp      # snmp request port
snmp            161/udp      # snmp request port
snmp-trap       162/tcp      # snmp monitor trap port
snmp-trap       162/udp      # snmp monitor trap port
```

4.3.2 SNMP Configuration

To let the SNMP daemon know what the trap destination will be, you have to check that the IP address or hostname of the SNMP listener is correct in the /etc/snmpd.conf file. When the adapter is on the monitoring host, the trap destination should be the local loopback address of the control workstation, as shown in the following example:

```
trap        public       127.0.0.1       1.2.3   fe      # loopback
```

SNMP Adapter on T/EC Server Node

Although it is not mentioned in the Tivoli documentation, it is possible to install the T/EC adapter on the T/EC server node and to send the SNMP traps via the network to the adapter on the server node. In that case, you have to change the trap destination address to the IP address of the T/EC server, as in the following example:

```
trap        public       9.12.1.38       1.2.3   fe      #risc38
```

If the /etc/snmpd.conf file is changed, it is necessary to refresh the snmpd daemon to re-read the configuration file. The snmpd daemon is a subsytem controlled by the System Resource Controller (SRC). A refresh of the daemon can be done by executing *refresh -s snmpd*.

4.3.3 Adapter Configuration

The SNMP event adapter has several files which are located in the /etc/Tivoli/tecad/etc directory. Some of these files need configuration changes before you can start the adapter.

4.3.3.1 The tecad_snmp.conf file

The tecad_snmp.conf file is the adapter configuration file. For the server location, the ServerLocation keyword needs to be added to let the adapter know which system the T/EC server node is.

```
ServerLocation=@risc38.itso.ibm.com
```

4.3.3.2 The tecad_snmp.oid file

One of the basic elements of SNMP is the Management Information Base (MIB), which indicates the type of information SNMP can contain. For the RS/6000 SP, three special groups of MIB are available:

- ibmSPconfig (1.3.6.1.4.1.2.6.117.1) contains SP system configuration information.
- ibmSPErrlogVars (1.3.6.1.4.1.2.6.117.2) contains information about the latest error log write that caused an SNMP trap.
- ibmSPEMVariables (1.3.6.1.4.1.2.6.117.3) contains information about the last Event Management event that caused an SNMP trap.

For more information about SNMP SP MIB, see *RS/6000 SP High Availability Infrastructure* or the *IBM PSSP for AIX Administration Guide*. A listing of the MIBs provided with PSSP 2.2 is included in Appendix B, “The SP MIBs” on page 147.

The tecad_snmp.oid file contains a mapping of MIB values to variable names which can be referenced in the tecad_snmp.cds file. In the example used, the SNMP traps generated for Event Management events are forwarded. We have added the object identifiers contained in Event Management traps to this tecad_snmp.oid file. The tecad_snmp.oid file contains a mapping of MIB values to variable names. These variables can be referenced in the tecad_snmp.cds file.

```
# IBM SP
#
"ibmSPEMEventID"                        "1.3.6.1.4.1.2.6.117.3.1.1"
"ibmSPEMEventFlags"                     "1.3.6.1.4.1.2.6.117.3.1.2"
"ibmSPEMEventTime"                      "1.3.6.1.4.1.2.6.117.3.1.3"
"ibmSPEMEventLocation"                  "1.3.6.1.4.1.2.6.117.3.1.4"
"ibmSPEMEventPartitionAddress"          "1.3.6.1.4.1.2.6.117.3.1.5"
"ibmSPEMEventVarsTableName"             "1.3.6.1.4.1.2.6.117.3.1.6"
"ibmSPEMEventVarsTableInstanceID"       "1.3.6.1.4.1.2.6.117.3.1.7"
"ibmSPEMEventVarName"                   "1.3.6.1.4.1.2.6.117.3.1.8"
"ibmSPEMEventVarValueInstanceVector"    "1.3.6.1.4.1.2.6.117.3.1.9"
"ibmSPEMEventVarValuesTableInstanceID"  "1.3.6.1.4.1.2.6.117.3.1.10"
"ibmSPEMEventVarValue"                  "1.3.6.1.4.1.2.6.117.3.1.11"
"ibmSPEMEventPredicate"                 "1.3.6.1.4.1.2.6.117.3.1.12"
```

4.3.3.3 The tecad_snmp.cds file

The T/EC server is using an event class structure to recognize events. These event classes are described in the tecad_snmp.cds file for the adapter and the tecad_snmp.baroc file for the T/EC server.

Both files are related to each other. The file tecad_snmp.cds defines how events are constructed from information contained in the SNMP trap. Normally, only default SNMP traps are defined in these files. Therefore, SP events generated by Event Perspectives are not recognized by the adapter and will not be forwarded or even recognized by the T/EC server. To make this all work, the specific SP event definitions need to be defined in these files.

For each event class the SNMP adapter supports, one or more class definition statements (CDS) have to be added to the configuration file to define which incoming events map to that class and how the slots of the outgoing event instance are to be filled. Figure 12 on page 49 shows an example of the class entry for one type of SP event.

A complete example of a Class Definition Statement file used with our SNMP adapter is listed in Appendix E, “Contents of SNMP Adapter Class Definition Statements” on page 183. Each trapid listed in the following description list is matched with a Class Definition in the TME 10 Enterprise Console.

In our example we used the convention of setting the trapid to indicate the severity level of the event. The following mapping was used:

trapid=1 SP_SNMP_FATAL_TRAP

trapid=2 SP_SNMP_CRITICAL_TRAP

trapid=3 SP_SNMP_MINOR_TRAP

trapid=4 SP_SNMP_WARNING_TRAP

trapid=5 SP_SNMP_HARMLESS_TRAP

trapid=?? SP_SNMP_UNKNOWN_TRAP

All ibmSPEM* values in the MIB are set and passed to T/EC. The corresponding BAROC file contains the same classes and variables.

You can add your own specific definitions to this example by copying one of the existing classes and placing it *before* the SP_SNMP_UNKNOWN_TRAP class.

Change the $SPECIFIC selection test to contain the value of your unique trapid as set in the PSSP event. Make the required changes to the BAROC file that is loaded into T/EC. Refer to Figure 12 as an example.

```
CLASS SP_SNMP_FATAL_TRAP
  SELECT
    1: ATTR(=,$ENTERPRISE),
           VALUE(PREFIX, "1.3.6.1.4.1.2.6.117") ;
    2: $SPECIFIC = 1;
    3: ATTR(=, "ibmSPEMEventID" ) ;
    4: ATTR(=, "ibmSPEMEventFlags" ) ;
    5: ATTR(=, "ibmSPEMEventTime" ) ;
    6: ATTR(=, "ibmSPEMEventLocation" ) ;
    7: ATTR(=, "ibmSPEMEventPartitionAddress" ) ;
    8: ATTR(=, "ibmSPEMEventVarsTableName" ) ;
    9: ATTR(=, "ibmSPEMEventVarsTableInstanceID" ) ;
   10: ATTR(=, "ibmSPEMEventVarName" ) ;
   11: ATTR(=, "ibmSPEMEventVarValueInstanceVector" ) ;
   12: ATTR(=, "ibmSPEMEventVarValuesTableInstanceID" ) ;
   13: ATTR(=, "ibmSPEMEventVarValue" ) ;
   14: ATTR(=, "ibmSPEMEventPredicate" ) ;
  FETCH
    1: IPNAME($SOURCE_ADDR) ;
  MAP
    hostname = $F1 ;
    enterprise = $ENTERPRISE ;
    pssp_EMEventID = $V3 ;
    pssp_EMEventFlags = $V4 ;
    pssp_EMEventTime = $V5 ;
    pssp_EMEventLocation = $V6 ;
    pssp_EMEventPartitionAddress = $V7 ;
    pssp_EMEventVarsTableName = $V8 ;
    pssp_EMEventVarsTableInstanceID = $V9 ;
    pssp_EMEventVarName = $V10 ;
    pssp_EMEventVarValueInstanceVector = $V11 ;
    pssp_EMEventVarValuesTableInstanceID = $V12 ;
    pssp_EMEventVarValue = $V13 ;
    pssp_EMEventPredicate = $V14 ;
END
```

Figure 12. The tecad_snmp.cds file. Class Definition Example for FATAL Traps.

The SELECT code segment of the class definition must contain one or more test statements that an incoming event must satisfy to match this specific class.

In this example, the $ENTERPRISE needs to map with the specific SP MIB, and the $SPECIFIC (that is, the trapid) needs to have the value 1. The event is then recognized as FATAL based on the trapid convention used when creating the SNMP event.

$SPECIFIC is the specific trapid which can be configured on the SP in Event Perspectives. This trapid is now used to forward the severity of the event that was sent by the SNMP trap. In this example six class definition entries were defined, to map to the six severity levels supported by T/EC.

1 FATAL

2 CRITICAL

3 MINOR

4 WARNING

5 HARMLESS

other UNKNOWN (For errors that did not match with this field)

The MAP code segment contains map statements to place the SNMP variable values in to corresponding BAROC variables defined in the tec_snmp.baroc file. These variables will be set to the values which are described in the SELECT field.

A complete listing of the SP-specific entries is shown in Appendix E, “Contents of SNMP Adapter Class Definition Statements” on page 183.

4.3.3.4 The tecad_snmp.baroc file

For every event definition added or changed in the tecad_snmp.cds file, the corresponding event class definition in the tecad_snmp.baroc file also needs to be added or changed to recognize the forwarded SNMP trap information of the adapter. Event definition content and syntax are described in the *TME 10 EIF User's Guide*.

Figure 13 on page 52 shows the event definitions added to the tecad_snmp.baroc file which was installed in /etc/Tivoli/tecad/etc on the Tivoli T/EC server.

The following classes have been defined to recognize SNMP traps that have been generated by the PSSP Event Manager and forwarded by your SNMP adapter. All ibmSPEM* data values set in the MIB have been translated to variables. These values appear in the message displayed on the T/EC console.

SP_SNMP_TRAP	a base class containing all of the BAROC variable definitions for the MIB values
SP_SNMP_FATAL_TRAP	flags trap as FATAL
SP_SNMP_CRITICAL_TRAP	flags trap as CRITICAL
SP_SNMP_MINOR_TRAP	flags trap as MINOR
SP_SNMP_WARNING_TRAP	flags trap as WARNING
SP_SNMP_HARMLESS_TRAP	flags trap as HARMLESS
SP_SNMP_UNKNOWN_TRAP	flags trap as UNKNOWN

These classes have corresponding definitions in the SNMP adapter tecad_snmp.cds file previously described in 4.3.3.3, “The tecad_snmp.cds file” on page 48. The SP_SNMP_TRAP class is the base class for SNMP events. This

means that the characteristics of this class are inherited by the underlying classes.

```
TEC_CLASS :
        SP_SNMP_TRAP ISA Specific_SNMP_Trap
        DEFINES {
                origin: dup_detect = yes;
                pssp_EMEventID:  INT32;
                pssp_EMEventFlags: INT32;
                pssp_EMEventTime:  INT32;
                pssp_EMEventLocation: INT32;
                pssp_EMEventPartitionAddress: STRING;
                pssp_EMEventVarsTableName: STRING;
                pssp_EMEventVarsTableInstanceID:  STRING;
                pssp_EMEventVarName:  STRING;
                pssp_EMEventVarValueInstanceVector: STRING;
                pssp_EMEventVarValuesTableInstanceID: STRING;
                pssp_EMEventVarValue:  STRING;
                pssp_EMEventPredicate: STRING;
                };
END
TEC_CLASS :
        SP_SNMP_FATAL_TRAP ISA SP_SNMP_TRAP
        DEFINES {
                severity: default = FATAL;
                };
END
TEC_CLASS :
        SP_SNMP_CRITICAL_TRAP ISA SP_SNMP_TRAP
        DEFINES {
                severity: default = CRITICAL;
                };
END
TEC_CLASS :
        SP_SNMP_MINOR_TRAP ISA SP_SNMP_TRAP
        DEFINES {
                severity: default = MINOR;
                };
END
TEC_CLASS :
        SP_SNMP_WARNING_TRAP ISA SP_SNMP_TRAP
        DEFINES {
                severity: default = WARNING;
                };
END
TEC_CLASS :
        SP_SNMP_HARMLESS_TRAP ISA SP_SNMP_TRAP
        DEFINES {
                severity: default = HARMLESS;
                };
END
TEC_CLASS :
        SP_SNMP_UNKNOWN_TRAP ISA SP_SNMP_TRAP
        DEFINES {
                severity: default = UNKNOWN;
                };
END
```

Figure 13. The tecad_snmp.baroc file, SP-related entries

The T/EC server needs to be prepared to receive the SNMP traps. You have to create a new source on the T/EC server with the name of SNMP. This process is well described in *TME 10 Cookbook for AIX Systems Management and Networking Applications* and *TME 10 Enterprise Console User's Guide*.

In summary, the steps involved are as follows:

1. Create a new rule base that will be your working rule base.
2. Copy the default rule base into your working rule base.
3. Copy the special T/EC class definition file (the BAROC file) from the distribution disk (for example on the control workstation) to the T/EC server. The class definition file is called tecad_snmp.baroc and it is installed in the directory /etc/Tivoli/tecad/etc.
4. Update the BAROC file with SP_SNMP adapter customizations.
5. Compile the rule base.
6. Load the rule base.
7. Stop and restart the event server.

After any change to the BAROC file, it has to be re-loaded into the rule base. Compile the rule base and load it into the server. Then stop and restart the server again.

In order to receive events in a T/EC console, an event group needs to be defined. Also, to receive events in that event group, a filter has to be specified as to what events should be directed to the event group.

There is a command available to simulate an event to test if the BAROC file is loaded correctly. To test the SP-specific entries the following example can be used:

```
wpostemsg -m "test from wpostemsg" SP_SNMP_FATAL_TRAP SNMP
```

4.3.4 Starting and Testing the Event Adapter

By default, the event adapter is always started when the control workstation is booting up. When you start the adapter for the first time, it is called a *cold start*. This can be done manually with the following command:

```
tecad_snmp -c /etc/Tivoli/tecad/etc/tecad_snmp.conf
```

Restarting a running adapter due to changes in the tecad_snmp.cds file is called a *warm start*. This can be done by using the `kill` command, as follows:

```
kill -HUP <process_id>
or
kill -1 <process_id>
```

During adapter testing, it can be very useful to have extra information about the functioning of the adapter. You can adjust the tracing options in the /etc/Tivoli/tecad/etc/tecad_snmp.err file. By default all trace information is sent

to /dev/null. Change /dev/null to a filename and you can follow the functioning of the adapter on the control workstation. For more information about the startup of the adapter, see *TME 10 Enterprise Console Event Adapter Guide: SNMP*.

The `wtdumprl` command can be used to investigate all incoming T/EC events. It will generate a report of events received.

The adapter and T/EC server are now ready to receive SNMP traps generated by PSSP Event Perspectives. Figure 14 on page 55 shows the Event Perspectives Definition window. For the event action, the SNMP trapid was defined with a value of 4. This will result in a warning event on the T/EC console window when this SNMP trap is generated.

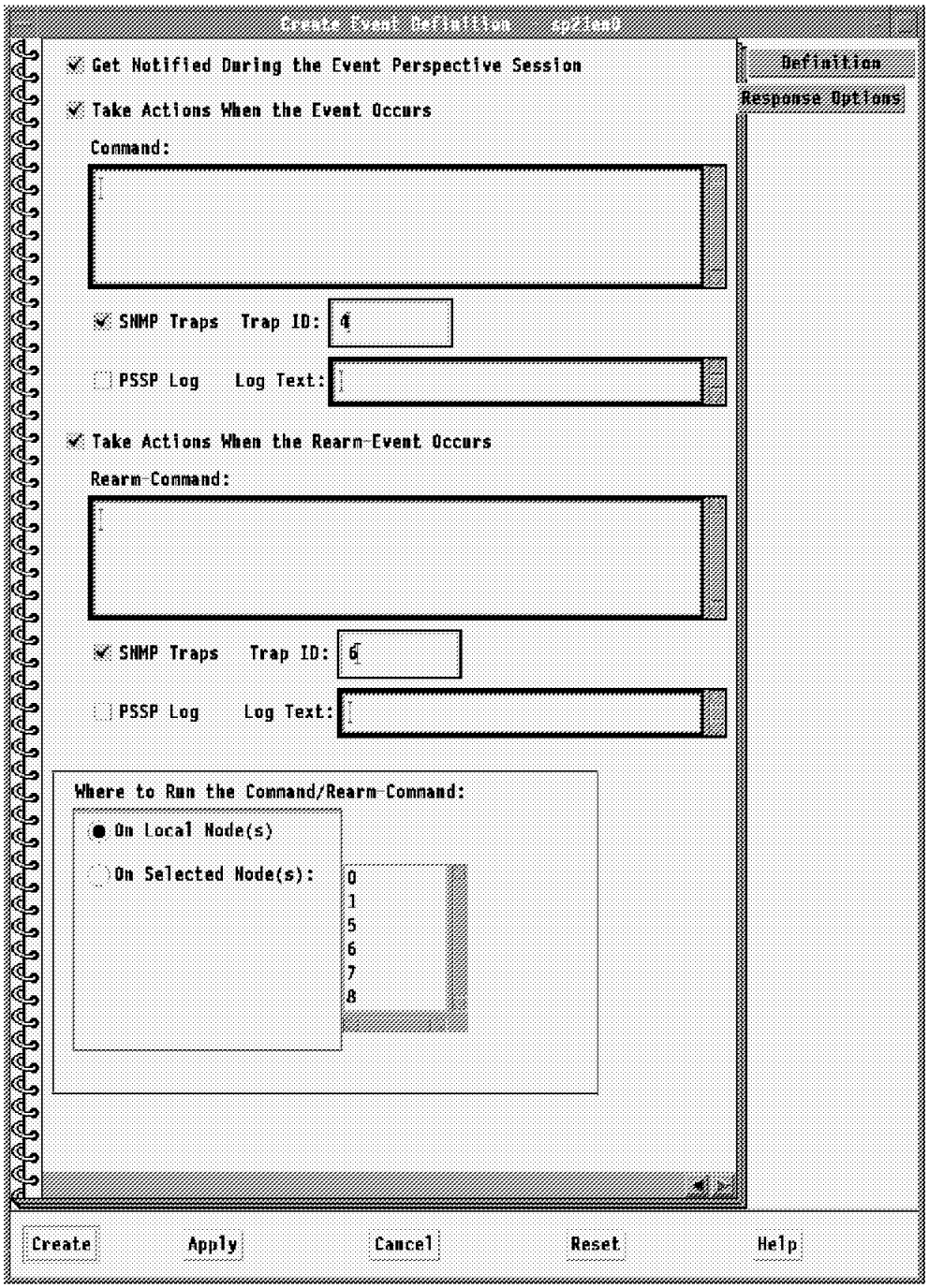


Figure 14. Perspective Event Definition: Example of a WARNING SNMP Trap for T/EC

Figure 15 on page 56 shows the T/EC event Console for SNMP trap messages. Here you can see the WARNING message indicating that the /tmp filesystem on the control workstation is more than 90% used.

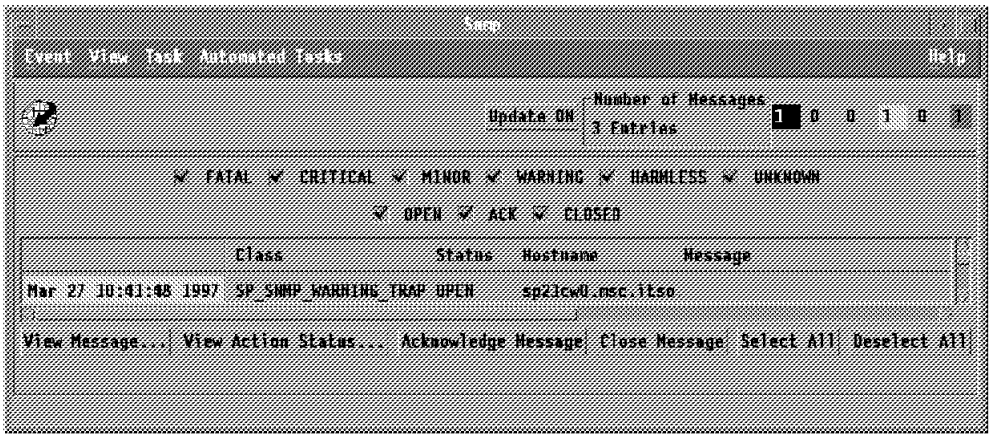


Figure 15. T/EC SNMP Event Console: SNMP trap WARNING message

Figure 16 on page 57 shows the T/EC Event Group Message Viewer with a message containing all available event information. You can see that all PSSP event variables are forwarded to T/EC.

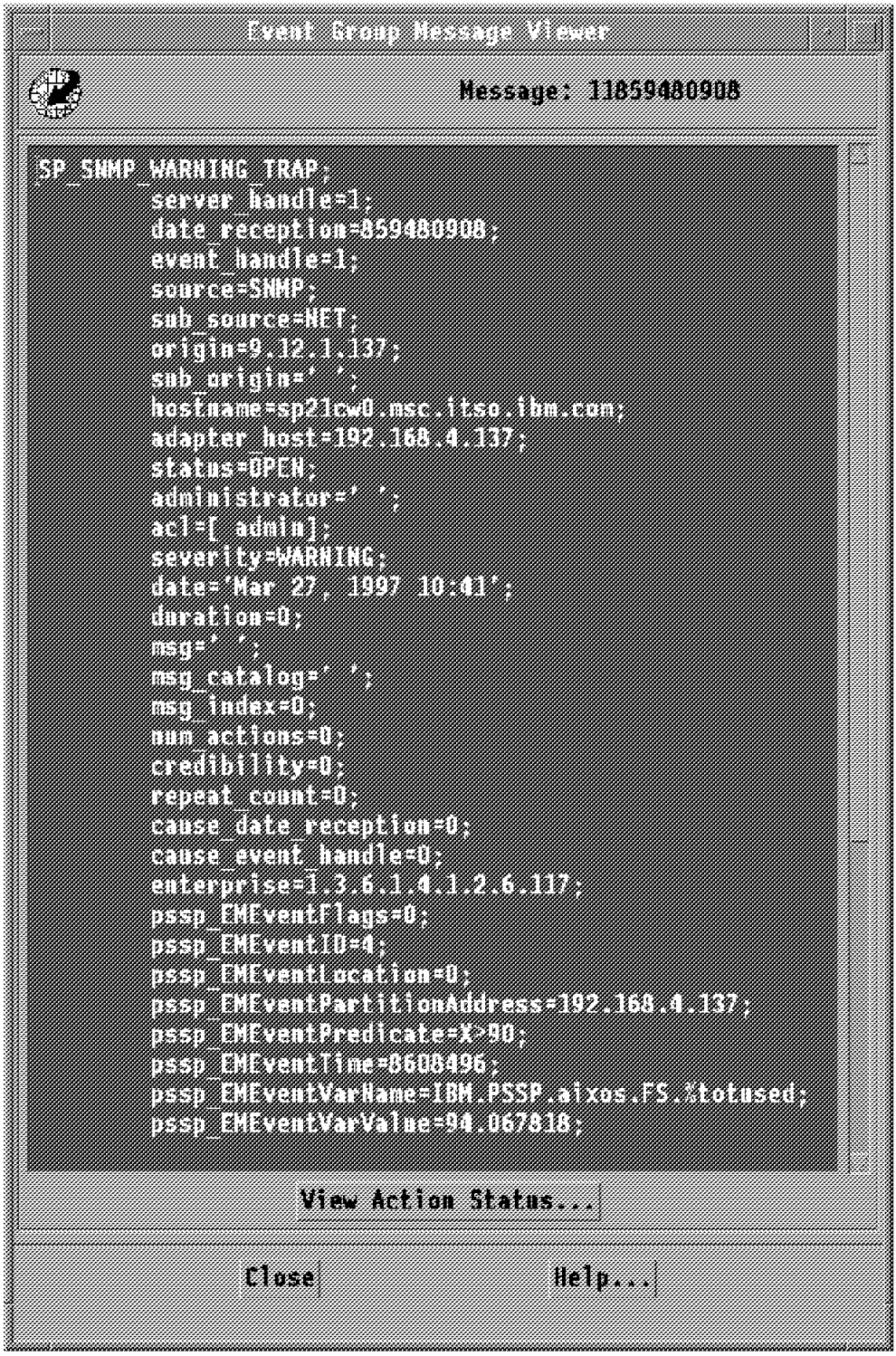


Figure 16. T/EC Message Viewer Window. T/EC message information describing the PSSP event that was forwarded by the SNMP trap. (*Note that this is an incomplete view. More data values are scrolled off the bottom.*)

With this information, it is possible to take corrective actions on the RS/6000 SP system. These corrective actions can be either automatically generated by T/EC or by manual intervention.

4.3.5 SNMP Adapter Value

The real advantage of using the SNMP adapter for forwarding SP events to the TME 10 Enterprise Console is that the administrator is able to use different trapid′s to assign severity levels, or any other distinguishing value, to an event. The severity levels or other values can be used to filter the way the events will appear in the TME 10 Enterprise Console, and what actions should be performed, when an event is generated.

In this chapter, the granularity is limited to six levels, where each level represents a severity level. Of course, a finer-grained level hierarchy could be implemented to reflect certain system aspects in the SP in the trapid. For example, the following method could be used:

trapid 10001 to 10005 can be used for switch-related traps.

trapid 11001 to 11005 can be used for SP hardware-related traps.

trapid 12001 to 12005 can be used for control workstation-related traps.

Moreover, this list could be extended in many directions. Of course when a finer granularity is implemented, the Class Definitions Statement and the BAROC file must be updated accordingly.

4.4 Using the TME 10 T/EC Logfile Adapter

One kind of general computing resource in the distributed environment is the log file, which documents the running of programs and processes. Each entry in the log file represents a confirmation of an activity, whether it was successful or not.

Log files are sometimes the only way to diagnose errors. However, it is not easy to get the necessary information from them. First you have to determine the location and filename of the log file. Also, the quantity of the information that systems and applications place in log files varies enormously. You may find yourself looking at a sequence of plain text messages, written one after another with no consideration for severity or meaning.

Some operating systems provide a general purpose logging mechanism that system components and applications can use. An example is the AIX syslogd daemon. With the T/EC logfile event adapter it is possible to generate T/EC events based on messages from syslogd, as well as from any log files on the system. The T/EC logfile event adapter (the default) provides a set of BAROC classes, imported into the rule base of the T/EC event server, to represent these messages. For details on the procedures to install and use the T/EC logfile event adapter, refer to the ITSO redbook, *TME 10 Cookbook for AIX Systems Management and Networking Applications*.

The sequence of actions is as follows:

1. The T/EC logfile event adapter analyzes the messages in the log file using pattern matching predefined in the Class Definition Statements configuration file, tecad_logfile.cds.
2. If the message matches a specific pattern/class entry, the event adapter transforms the message into an event and sends it to the T/EC event server. The default filters in the tecad_logfile.conf file are:
   - Logfile_Base
   - Logfile_Sendmail

- Amd_Unmounted
- Amd_Mounted

3. The T/EC event server then uses the BAROC file logfile.baroc with the T/EC logfile event adapter to process the event.

Most of the classes in the configuration files tecad_logfile.cds and tecad_logfile.baroc are predefined for general purpose solutions of system management. It may be necessary to create a customized solution for using the T/EC logfile event adapter to analyze the messages from specific applications, and then construct them into an event that can be processed by the T/EC event adapter.

The following sections describe an example of how to incorporate T/EC and the T/EC logfile event adapter with the SP PSSP Problem Management subsystem to perform system monitoring tasks. The basic concept of this example is that the PSSP Problem Management subsystem client code, which is run where a defined event occurs, uses the system syslog facility to pass a log file entry containing an SP-specific message to the T/EC logfile event adapter. The logfile adapter uses the customized logfile tecad_logfile.cds to match the message to a specific class and send it as an event to the T/EC Event Server, which uses a customized BAROC file to process the event and display it on the event console.

4.4.1 How to Forward Events from a Log File

The following steps document how an event defined by a client of the Problem Management subsystem on the control workstation can be translated to the T/EC console, using the logfile adapter. In this example, the SP-specific message in the log file entry contains either the word busy or idle. Based on the difference in the message, the logfile adapter is configured accordingly, such that the Class Definition Statements file (xyz.cdf file) filters the description field of the SP-specific log file entry.

The following steps were excecuted on the control workstation. Steps one to six represent the steps to define the parsing method and the format of the log file entry.

Step 1 Alter the configuration file of the system syslog daemon to save SP-specific messages generated by the PSSP Problem Management subsystem in a log file:

```
cat >> /etc/syslog.conf <<EOF
daemon.notice       /tmp/log_pssp.log
EOF
```

Step 2 Create the log file that will contain the SP-specific messages:

```
/usr/bin/touch /tmp/log_pssp.log
```

Step 3 Notify the system syslog daemon that the configuration file has been changed:

```
/usr/bin/refresh -s syslogd
```

Step 4 Use the PSSP Event Perspectives GUI or the pmandef command to define an event to monitor the CPU utilization (for this example) on the control workstation. In this example, an event will be generated when the CPU utilization on the control workstation falls below a predefined condition. The event definitions are shown in Figure 17 on

page 60. (For details on the procedure to define events, refer to *IBM PSSP for AIX Administration Guide*.

When the ratio of CPU idle time is less than twenty-five percent, Problem Management writes a PSSP log entry to the syslog file to indicate the CPU busy situation.

When the ratio of CPU idle time is greater than fifty percent, Problem Management writes a SP-specific message to the syslog file to indicate the CPU idle situation.

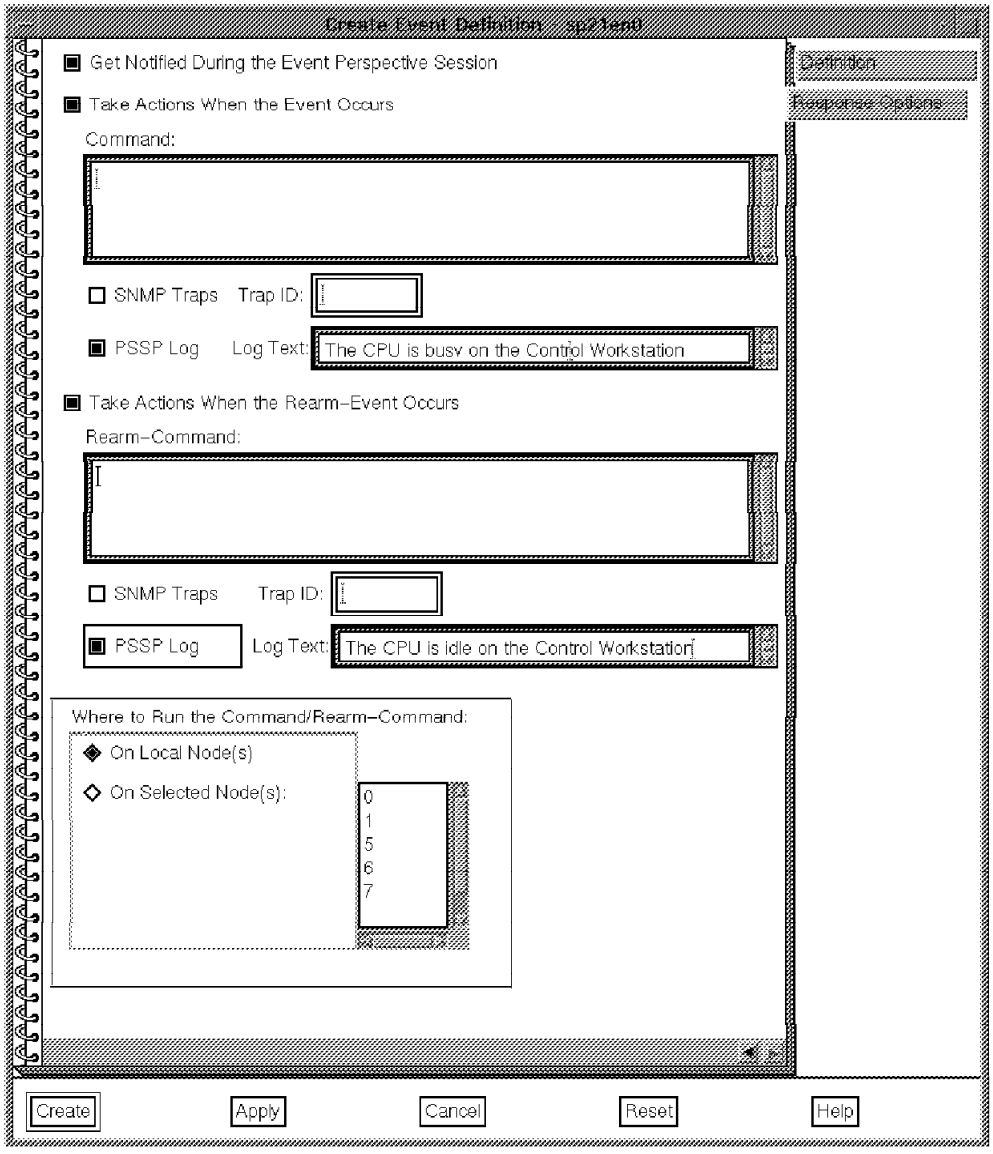


Figure 17. Event Definition. This event definition is created with the Event Management Perspectives.

The equivalent pmandef command would look like the following:

```
pmandef -s LOG_PSSP_CPU
        -e "IBM.PSSP.aixos.cpu.%idle:CPU=cpu0;NodeNum=0;X<25"
        -l "The CPU is busy on the control workstation"
        -r "X>50"
        -L "The CPU is idle on the control workstation"
```

Step 5 Analyze the string structure of the following two log file entries containing the SP-specific messages written by Problem Management client code which monitors CPU utilization in the log file /tmp/log_pssp.log. The purpose of this step is to fetch SP-specific elements from each log file entry and map them into slots in a BAROC event format. The log file entry for a high-usage CPU in this example looks like the following:

```
Mar 20 14:31:23 sp21en0 pmand[39444]: SP Problem Mgmt:
Monitored Situation Exists:
Name=LOGFILE_PSSP_CPU
Node Number=0
Resource Variable=IBM.PSSP.aixos.cpu.%idle
Instance Vector=CPU=cpu0;NodeNum=0
Predicate=X<25
Description=The CPU is busy on control workstation!
```

The log file entry for a low-usage CPU in this example looks like the following:

```
Mar 20 14:32:08 sp21en0 pmand[39444]: SP Problem Mgmt:
Monitored Situation Cleared: Name=LOGFILE_PSSP_CPU
Node Number=0
Resource Variable=IBM.PSSP.aixos.cpu.%idle
Instance Vector=CPU=cpu0;NodeNum=0
Predicate=X>50
Description=The CPU is idle on control workstation!
```

Step 6 Remove the entry added in Step 1 from the configuration file of system syslog daemon /etc/syslog.conf:

```
vi /etc/syslog.conf
```

In steps seven to twenty-two, the operating environment is defined and set up. This includes configuring the event group in T/EC.

Step 7 Notify the system syslog daemon that the configuration file has been changed:

```
/usr/bin/refresh -s syslogd
```

Step 8 On the T/EC Event Server, design and code the BAROC file, named log_pssp.baroc, according to the analysis results from Step 5, as shown in the following example:

```
cat > log_pssp.baroc <<EOF
# Component        : log_pssp.baroc
#

TEC_CLASS :
Log_PSSP ISA Logfile_Base
DEFINES {
                pssp_daemon: STRING, default="pmand";
};
END

TEC_CLASS :
Log_PSSP_CPU ISA Log_PSSP
DEFINES {
                pssp_EM_event_name: STRING, default="N/A";
                pssp_node_number: STRING, default="N/A";
                pssp_EM_resource_variable: STRING, default="N/A";
                pssp_EM_IV_cpu: STRING, default="N/A";
                pssp_EM_IV_node: STRING, default="N/A";
                pssp_EM_predicate: STRING, default="N/A";
                pssp_EM_discription: STRING, default="N/A";
};
END

TEC_CLASS :
Log_PSSP_CPU_High ISA Log_PSSP_CPU
DEFINES {
severity: default = CRITICAL;
};
END

TEC_CLASS :
Log_PSSP_CPU_Low ISA Log_PSSP_CPU
DEFINES {
severity: default = HARMLESS;
};
END
EOF
```

Step 9 Define a new T/EC event source type, a log file labeled syslog, for the log file events.

> **Note:**
>
> It is recommended that the TME 10 Enterprise Console desktop be used to perform all steps relative to T/EC.

Step 10 Create a new T/EC event group that selects log file events to be displayed.

Step 11 Assign the T/EC event group to the event console of the appropriate administrator.

Step 12 Create a new T/EC rule base. Copy all rules and classes of the default rule base into the new rule base.

Step 13 Import log_pssp.baroc into this rule base. Its position in the class list must be behind tecad_logfile.baroc.

Step 14 Compile and reload the rule base.

Step 15 Stop and start the T/EC event server.

Step 16 Copy tecad_logfile.fmt into the /tmp directory:

```
cd /tmp
cp /etc/Tivoli/tecad/etc/tecad_logfile.fmt tecad_logfile.fmt
```

Step 17 Copy /tmp/log_pssp.log from the control workstation into the local /tmp directory:

```
cd /tmp
rcp sp21en0:/tmp/log_pssp.log /tmp/log_pssp.log
```

Step 18 Use the T/EC log file configuration facility to translate and add the two SP-specific messages to /tmp/tecad_logfile.fmt, as follows:

> **Note:**
>
> For details on the procedures to install and use the T/EC logfile configuration facility, refer to the *TME 10 Enterprise Console User's Guide*.

- Click on the event server icon with the right button to start the logfile configuration facility, as shown in Figure 18.

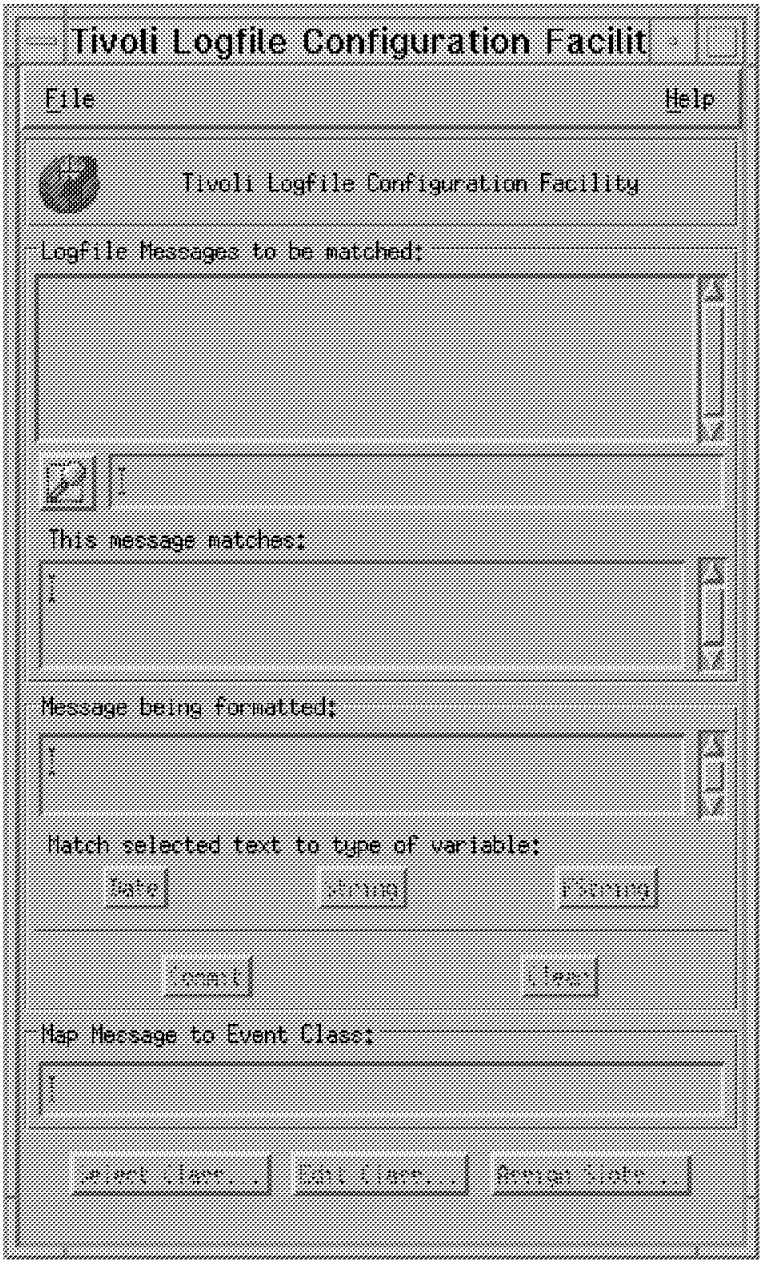


Figure 18. Using the Logfile Configuration Facility. Starting the LCF.

- Select **File→Open Format** and select **/tmp/tecad_logfile.fmt**.
- Select **File→Open Classes** and select **Rule Base** as your rule base.
- Select **File→Open Logfile** and select **/tmp/log_pssp.log**.

  Now the dialog of the logfile configuration facility will look like Figure 19.

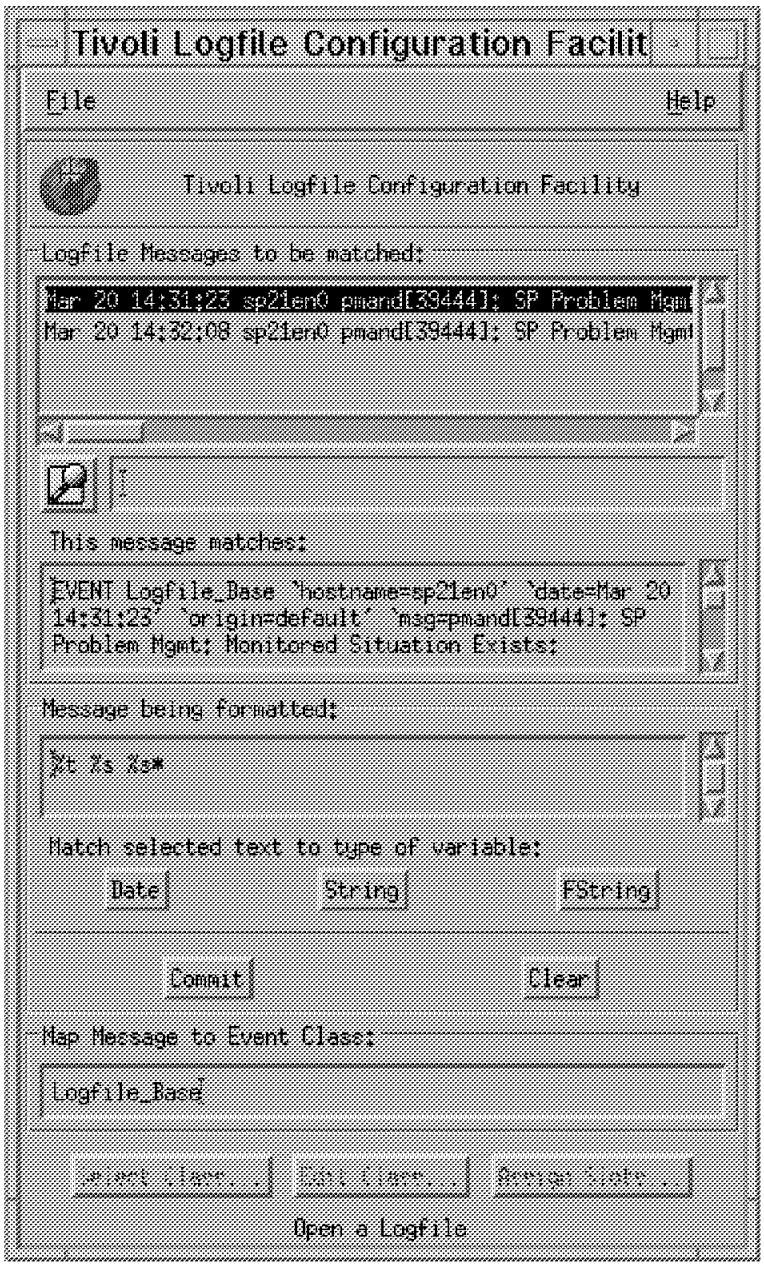


Figure 19. The Logfile Configuration Facility Dialog

- Click on the first SP-specific message in the dialog and look at its current mapping.
- Click **Clear** to start a new mapping. When finished, click **Commit**.
- Click **Select Class...**, then choose the **Log_PSSP_CPU_High** class, as shown in Figure 20 on page 65.

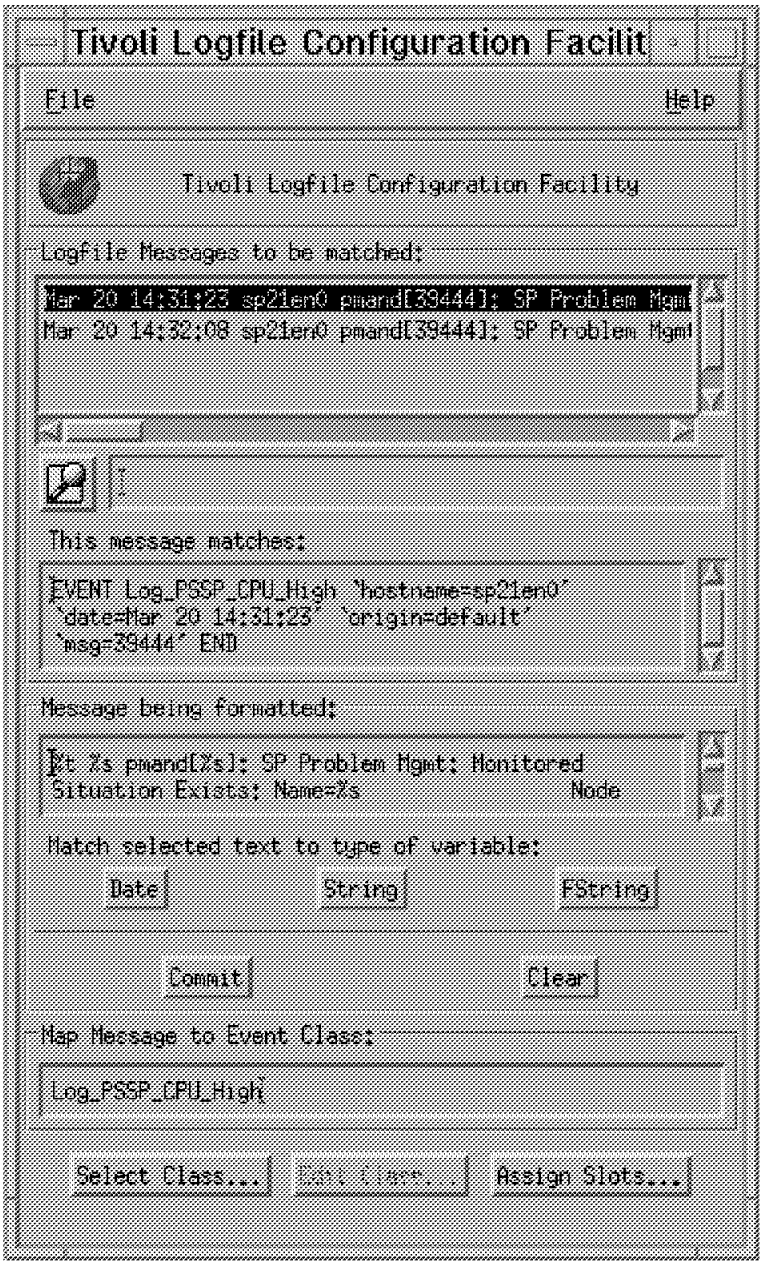


Figure 20. Selecting the Class with the Logfile Configuration Facility

- Click **Assign Slots...** to map the string structure of the message into BAROC of the Log_PSSP_CPU_High class, as shown in Figure 21 on page 66.

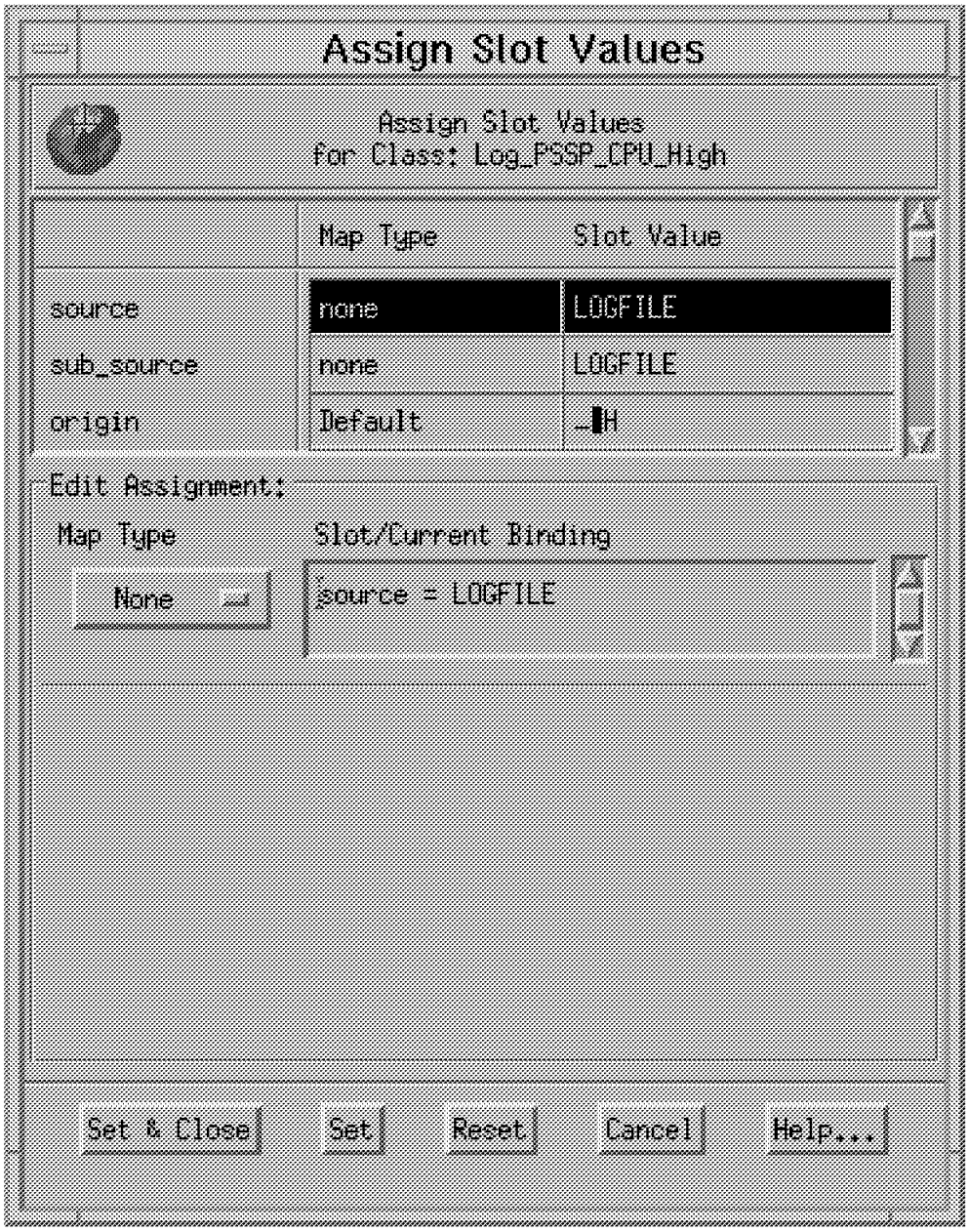


Figure 21. Mapping the Structure into the BAROC File

- Click **Set & Close** to save the results. Repeat for the second message, but map it to the Log_PSSP_CPU_Low class.

The following lines are added to /tmp/tecad_logfile.fmt by the T/EC logfile configuration facility after you have finished this step.

```
FORMAT Log_PSSP_CPU_High FOLLOWS Logfile_Base
%t %s pmand[%s]: SP Problem Mgmt: Monitored Situation Exists:
Name=%s Node Number=%s Resource Variable=%s Instance Vector=CPU=%s;
NodeNum=%s Predicate=%s Description=%s+ busy %s+
pid $3
pssp_EM_event_name $4
pssp_node_number $5
pssp_EM_resource_variable $6
pssp_EM_IV_cpu $7
pssp_EM_IV_node $8
pssp_EM_predicate $9
END

FORMAT Log_PSSP_CPU_Low FOLLOWS Logfile_Base
%t %s pmand[%s]: SP Problem Mgmt: Monitored Situation Exists:
Name=%s Node Number=%s Resource Variable=%s Instance Vector=CPU=%s;
NodeNum=%s Predicate=%s Description=%s+ idle %s+
pid $3
pssp_EM_event_name $4
pssp_node_number $5
pssp_EM_resource_variable $6
pssp_EM_IV_cpu $7
pssp_EM_IV_node $8
pssp_EM_predicate $9
END
```

Step 19 On the control workstation, copy /tmp/tecad_logfile.fmt from the T/EC server into the local /etc/Tivoli/tecad/etc directory:

```
cd /etc/Tivoli/tecad/etc
cp tecad_logfile.fmt tecad_logfile.fmt.orig
rcp risc38:/tmp/tecad_logfile.fmt tecad_logfile.fmt
```

Step 20 Generate a new Class Definition Statement configuration file for the T/EC logfile event adapter:

```
cp tecad_logfile.cds tecad_logfile.cds.orig
../bin/logfile_gencds tecad_logfile.fmt > tecad_logfile.cds
```

Step 21 Notify the T/EC logfile event adapter that the configuration file has been changed:

```
../bin/init.tecad_logfile stop
../bin/init.tecad_logfile start
```

Step 22 On the T/EC server, start up the event console of appropriate administrator.

Later you will see the two major log file events with the two SP-specific messages displayed in the event console, as shown in Figure 22 on page 68.

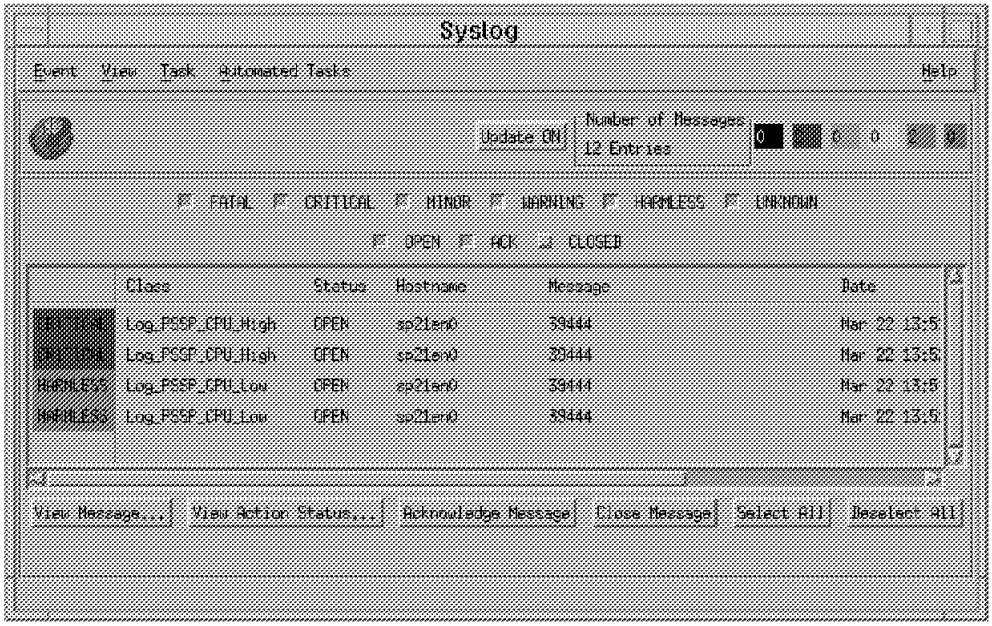


Figure 22. CPU Events Captured in T/EC

The information available in the T/EC console shows the availability of the Problem Management variables, as parsed to the generated event. Figure 23 on page 69 shows the detailed information of the event in the T/EC console.

Figure 23. Problem Management Information. Available information in the detailed view.

4.4.2 How to Assign Severities Using a Log File

There is another, similar way to use the T/EC logfile event adapter, which is to repeat all the steps of 4.4.1, “How to Forward Events from a Log File” on page 59, except for the use of the Event Management class definitions as filters in the logfile adapter.

In this example, the classes of the Event Management infrastructure are used to assign a different severity level when an event from that class is generated. This means that for the classes listed and defined in the SDR

(EM_Resource_Class), different severity levels are assigned. The following classes, available in a standard PSSP 2.2 installation, are assigned a matching severity:

Class	Severity
SP_HW	CRITICAL
aixos	MINOR
pm	CRITICAL
CSS	CRITICAL
HARMLD	CRITICAL
LL	CRITICAL
Membership	CRITICAL
PRCRS	CRITICAL
Prog	CRITICAL
Response	FATAL
VSD	MINOR

Important

Note, the assignments of severity levels in this example are arbitrary and may be completely different in other implementations. They are solely meant to show how to transfer information from the Problem Management subsystem to TME 10.

Similarly, the resource variable classes can be mapped to T/EC classes:

Class	TEC_CLASS
SP_HW	EM_SP_HW
aixos	EM_aixos
pm	EM_pm
CSS	EM_CSS
HARMLD	EM_HARMLD
LL	EM_LL
Membership	EM_Membership
PRCRS	EM_PRCRS
Prog	EM_Prog
Response	EM_Response
VSD	EM_VSD

Since the different classes of the Event Manager are now used to generate a different severity level in T/EC, a customized BAROC file should be created to reflect these options in the T/EC event server. The following is an example of a BAROC file for Event Management log entries:

```
#
#  Component    : log_pssp_ex2.baroc
#

TEC_CLASS :
         Log_EM ISA Logfile_Base
         DEFINES {
               severity: default = WARNING;
               pssp_daemon: STRING, default="pmand";
               PMAN_HANDLE: STRING;
               PMAN_IVECTOR: STRING;
               PMAN_LOCATION: STRING;
               PMAN_PRED: STRING;
               PMAN_PRINCIPAL: STRING;
               PMAN_RVCOUNT: INTEGER;
               PMAN_RVNAME: STRING;
               PMAN_RVTYPE: STRING;
               PMAN_TIME: STRING;
               PMAN_RVFIELD0: STRING;
               PMAN_RVFIELD1: STRING;
               PMAN_RVFIELD2: STRING;
               PMAN_RVFIELD3: STRING;
               PMAN_RVFIELD4: STRING;
               PMAN_RVFIELD5: STRING;
               PMAN_RVFIELD6: STRING;
               PMAN_RVFIELD7: STRING;
               PMAN_RVFIELD8: STRING;
               PMAN_RVFIELD9: STRING;
         };
END

TEC_CLASS :
         EM_SP_HW ISA Log_EM
         DEFINES {
              severity: default = CRITICAL;
         };
END

TEC_CLASS :
         EM_aixos ISA Log_EM
         DEFINES {
              severity: default = MINOR;
         };
END

TEC_CLASS :
         EM_pm ISA Log_EM
         DEFINES {
              severity: default = CRITICAL;
         };
END

TEC_CLASS :
         EM_CSS ISA Log_EM
         DEFINES {
              severity: default = CRITICAL;
         };
END
```

```
TEC_CLASS :
         EM_HARMLD ISA Log_EM
         DEFINES {
              severity: default = CRITICAL;
         };
END

TEC_CLASS :
         EM_LL ISA Log_EM
         DEFINES {
              severity: default = CRITICAL;
         };
END

TEC_CLASS :
         EM_Membership ISA Log_EM
         DEFINES {
              severity: default = CRITICAL;
         };
END

TEC_CLASS :
         EM_PRCRS ISA Log_EM
         DEFINES {
              severity: default = CRITICAL;
         };
END

TEC_CLASS :
         EM_Prog ISA Log_EM
         DEFINES {
              severity: default = CRITICAL;
         };
END

TEC_CLASS :
         EM_Response ISA Log_EM
         DEFINES {
              severity: default = FATAL;
         };
END

TEC_CLASS :
         EM_VSD ISA Log_EM
         DEFINES {
              severity: default = MINOR;
         };
END
```

To expand this example, the variable number of Problem Management variables, when an SBS-type resource variable is parsed, is taken into consideration.

Note:

An SBS-type resource variable is part of the resource variable types defined in the Event Management infrastructure.

SBS stands for Structured Byte String and represents a one-dimensional array containing variables. These variables can have one or more fields. The number of fields is dependent on the resource monitor feeding the specific SBS.

For example, the resource variable IBM.PSSP.Prog.xpcount is a SBS-type resource variable, where a number of fields are provided. The first field contains the current program count, and the second field contains the previous program count. The variable type in each field can be of any type.

An SBS variable can have many entries. The T/EC console is not aware of this, so to prepare the T/EC, the format file for the logfile adapter should be configured such that all possibilities of messages are reflected.

In this example, we limited the number of SBS fields to one big field, containing all the PMAN_RVFIELD variables. Also note that each class in the BAROC definition should have a class definition statement. As an example, the following entry represents the class deinition statement for EM_Prog. Similar definitions should be made for all classes defined in the log_pssp_ex2.baroc file.

```
FORMAT EM_Prog FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.Prog.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END
```

A complete class definition statement file as used in this example can be found in Appendix F, “Logfile CDS” on page 189.

In order to get all PMAN_RVFIELD variables at the end of the log file message string, the Problem Management variables must be sorted before they are sent to the syslog daemon. At the writing of this document, a practical rather than elegant solution was developed to make that possible. The command to sort the Problem Management environment variables looks like this:

```
/usr/bin/env | /usr/bin/grep PMAN | sed 's/_RVFIELD/zzzz/'
             | sort | sed 's/zzzz/_RVFIELD'
```

To reflect this, the filter should be added to the command in the event response of the event definition. The event definition executes the logger command to send a log entry to the syslog daemon with all the Problem Management environment variables attached, as shown in the following example:

```
pmandef -s example2
        -e "AnyReourceVariable;AnyInstanceVector;AnyPredicate"
        -c "/usr/bin/env | /usr/bin/grep PMAN |
            sed 's/_RVFIELD/zzzz/' | sort | sed 's/zzzz/_RVFIELD' |
            /usr/bin/xargs /usr/bin/logger -pdaemon.notice"
        -r "AnyReArmPredicate"
        -C "/usr/bin/env | /usr/bin/grep PMAN |
            sed 's/_RVFIELD/zzzz/' | sort | sed 's/zzzz/_RVFIELD' |
            /usr/bin/xargs /usr/bin/logger -pdaemon.notice"
```

After successful configuration of the log file source and the event group, the event console is able to receive messages from the log file source. An example of such a window is displayed in Figure 24.

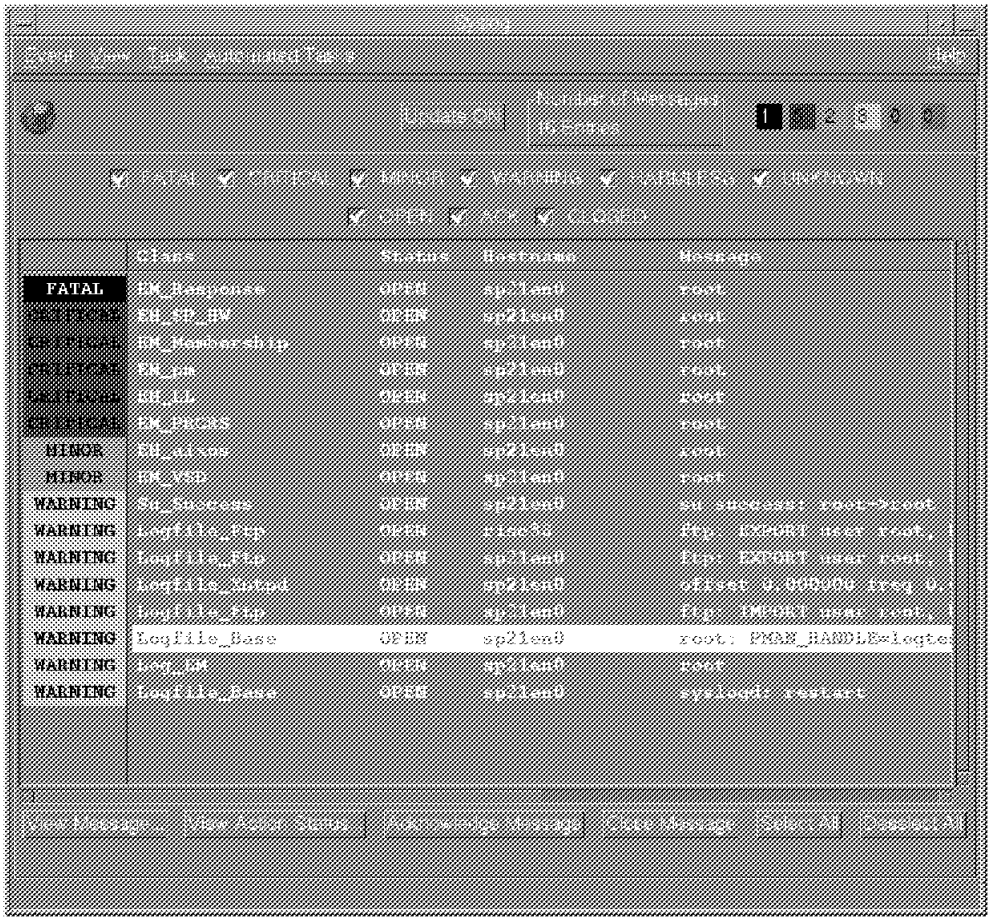


Figure 24. T/EC LOGFILE Source. Display of Events from Problem Management.

The advantage of the second example is the ability to get more of the Problem Management environment variables than the first one does. But through the customization capability of the T/EC logfile, the event adapter operates on non-pipe log files written by the syslog daemon. The first example is easily modified to perform SP system management tasks based on the other SP system log files, for example /var/adm/SPlogs/SPdaemon.log.

4.4.3 Summary of Using the Logfile Adapter

A large number of events can be captured with the logfile adapter.

The RS/6000 SP uses a variety of log files, which can be consolidated to the TME 10 Enterprise Console using the logfile adapter. As described in 4.4.1, “How to Forward Events from a Log File” on page 59 and 4.4.2, “How to Assign Severities Using a Log File” on page 69, there are a number of ways to interpret the contents of a log file entry. In both examples, the TME 10 Enterprise Console for log file events reads the contents of fields and, depending on the field characteristics, a certain event with a certain severity will be generated.

Of course the best solution for assigning severity levels to typical events is to make sure that the description field contains a string determining the severity of the event. So, the most flexible implementation of using the logfile adapter is to generate events from the Problem Management subsystem, where the log file entry contains a message indicating the severity of the event. This case was not investigated specifically for this residency, but would be easy to determine from the contents of this chapter. Obviously, the format file (tecad_logfile.fmt) file should be recreated and loaded in the TME 10 Enterprise Console, to reflect the changed implementation.

4.5 Using NetView/6000 for AIX and TME 10 Enterprise Console

The objective of this section is to send the SNMP traps from NetView/6000 for AIX to the TME 10 Enterprise Console. We created SP-specific events from PSSP 2.2, using the Problem Management component. These events will first go into NetView/6000 for AIX and be displayed on the Event Console. The event integration could possibly stop by using the NetView/6000 for AIX software only. In this section, we extended the possibilities and also forwarded the events to the T/EC. The T/EC rule engine will process the incoming event, display it on the Event Console, and initiate actions defined to respond to that type of event.

4.5.1 Environment Introduction

The versions used in this example are NetView/6000 for AIX (Ver. 4.1.2) and T/EC (Ver. 2.6) server running on systems named sp2en0 and risc38, respectively. We used the SP2 control workstation named sp21en0 to generate the SNMP traps from PSSP 2.2. Figure 25 on page 76 describes the different components that processed the events used in our lab.

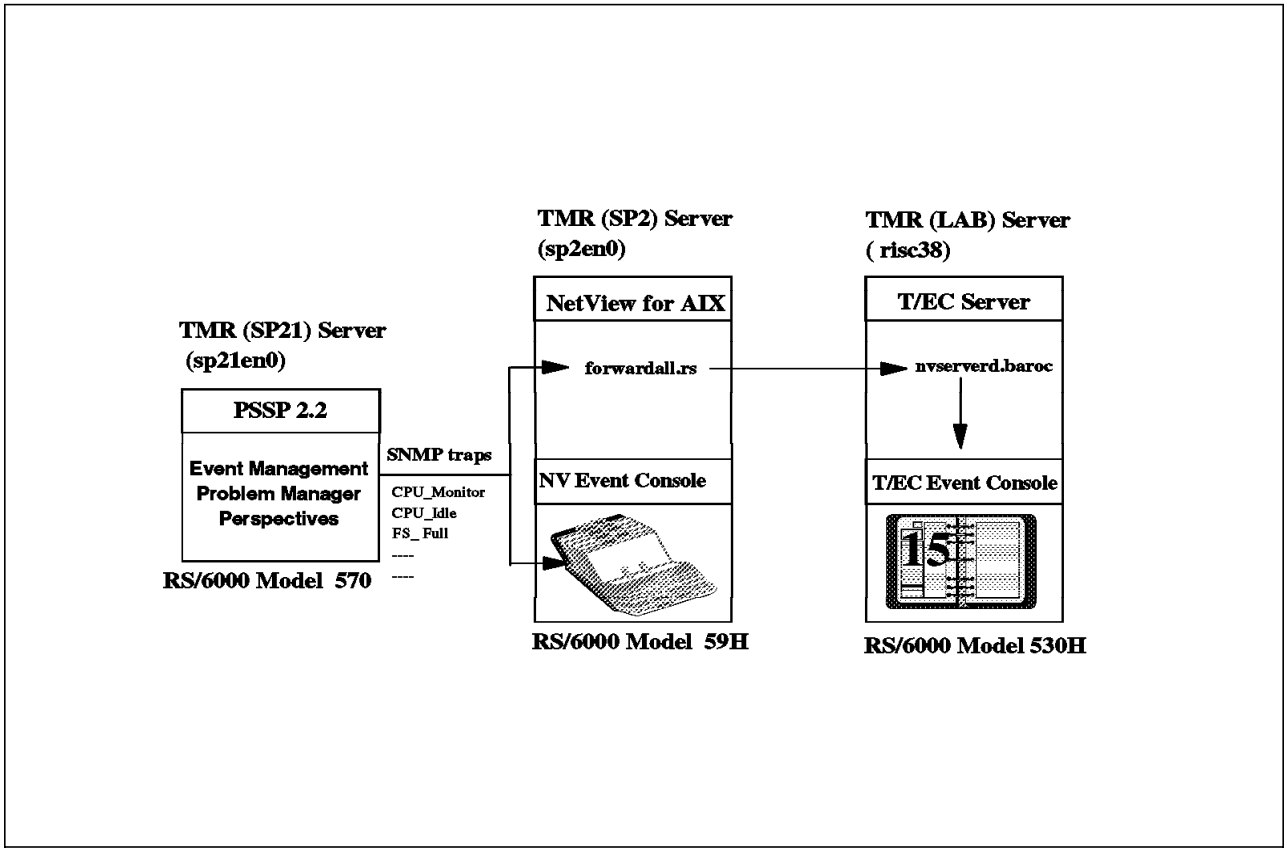


Figure 25. Event Consoles. The forward function of NetView/6000 for AIX.

4.5.2 Setup procedure in NetView/6000 for AIX

This section describes the procedure to configure NetView/6000 for AIX to be used as a forwarding mechanism to send messages to the T/EC.

Step 1 Load the SP MIBs.

The SP MIB located in /usr/lpp/ssp/config/snmp_proxy/ibmSPMIB.my file must be copied into the NetView running station under /usr/OV/snmp_mibs directory. The purpose of loading a MIB is to define the MIB objects so that NetView/6000 for AIX can understand those SP-specific MIB definitions.

The MIB must be loaded into NetView/6000 for AIX, using **Options→Load/Unload→SNMP...** from the NetView window.

Once the MIB is loaded, as shown in Figure 26 on page 77, you can traverse the MIB tree and select objects from the SP-specific MIB (Enterprise ID 1.3.6.1.4.1.2.6.117) by selecting **MIB Browser** in the Tools function menu to get or set the MIB values.

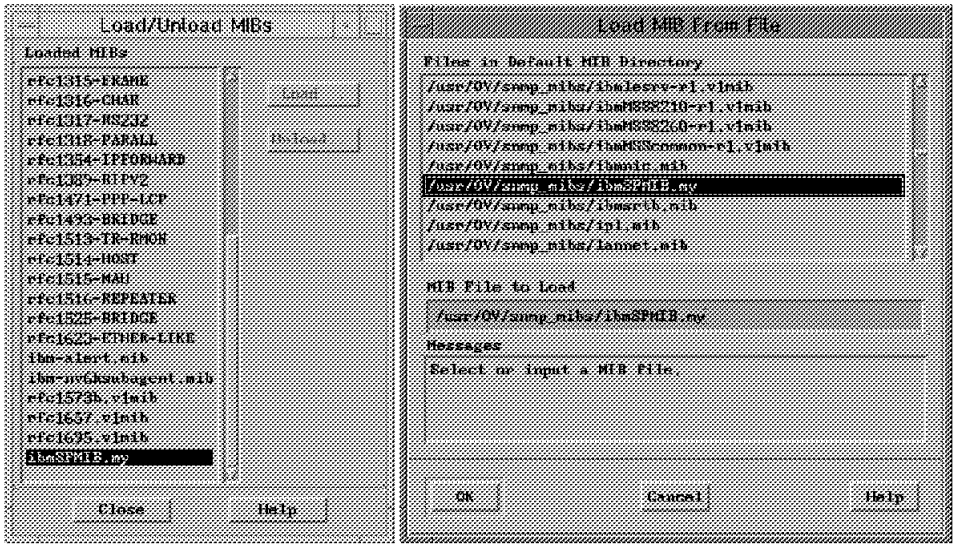


Figure 26. SP MIBS -- Loading the SP MIB

Figure 27 on page 78 shows the MIB browser as accessed from the NetView console.

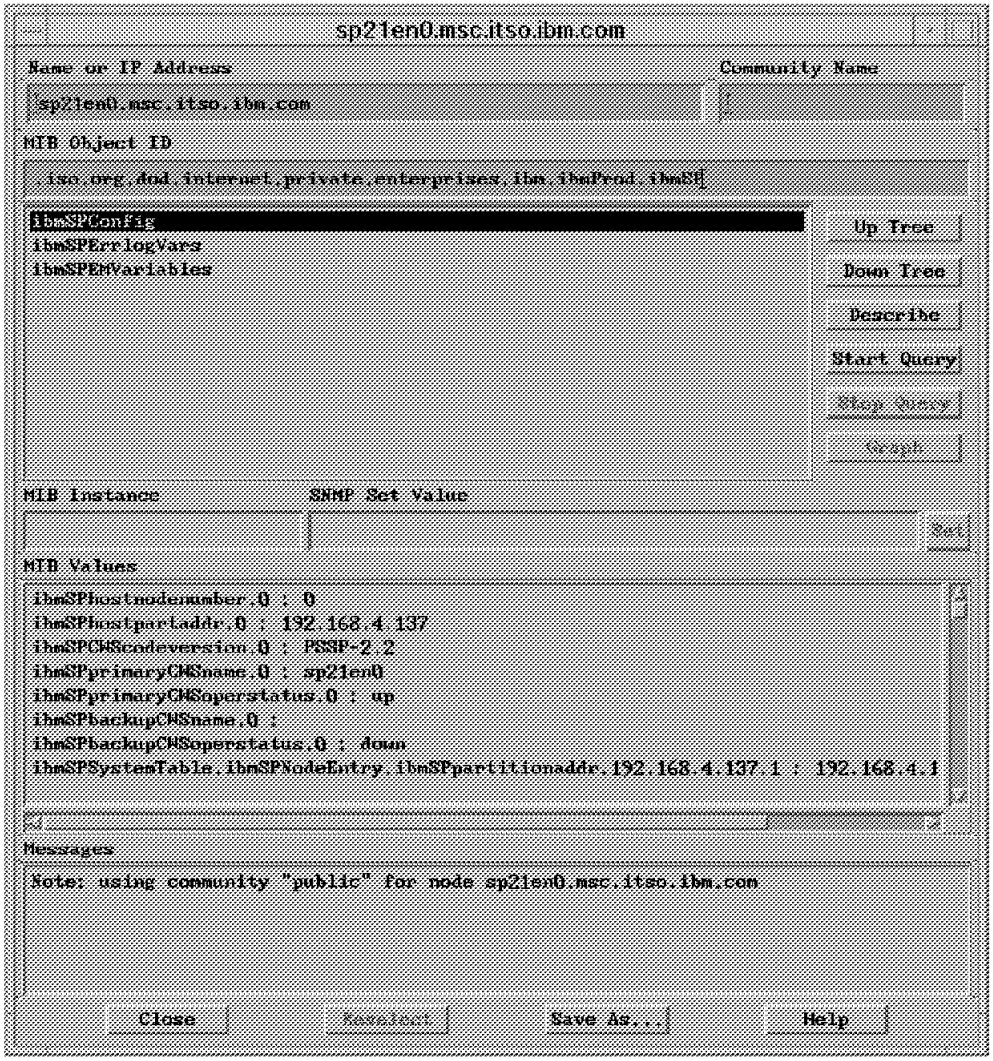


Figure 27. The NetView MIB Browser -- Viewing the SP MIB

Step 2 Customize Traps.

Every vendor has a MIB that describes device operations that can be performed. Vendors can specify traps that they expect to receive from agents that support their MIBs. To provide more specific MIB information for SP, first select **Options→Event Configuration→Trap Customization: SNMP** from the NetView menu item.

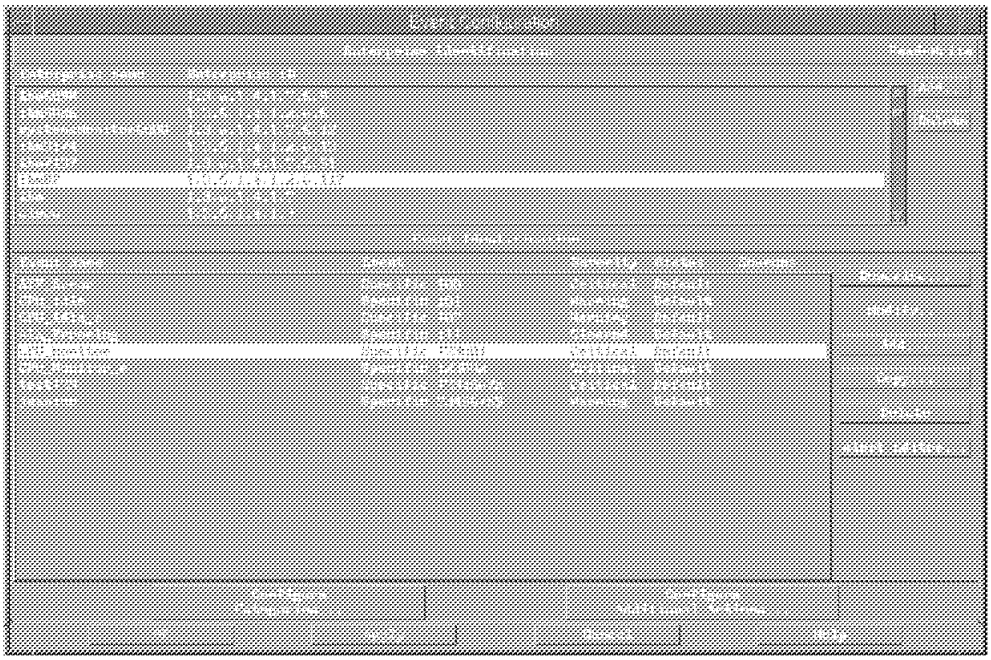


Figure 28. Event Configuration -- Register the SP-specific Traps

Next, select the **ibmSP** enterprise in the table and add the traps definition, using **Option→Event Configuration→Trap Customization: SNMP...** in NetView menu bar to configure the SP traps from the ibmSP category.

Notes:

SNMP trap is an unsolicited event from an agent to the SNMP manager. There are seven generic types:

1. Agent cold start
2. Agent warm start
3. Link down
4. Link up
5. Authentication failure (community name error)
6. EGP neighbor loss
7. Enterprise-specific defined traps

Enterprise-specific traps have additional identifiers:

1. Enterprise ID: object ID from the private entry of MIB tree
2. Specific Trap ID: defined by source of the trap

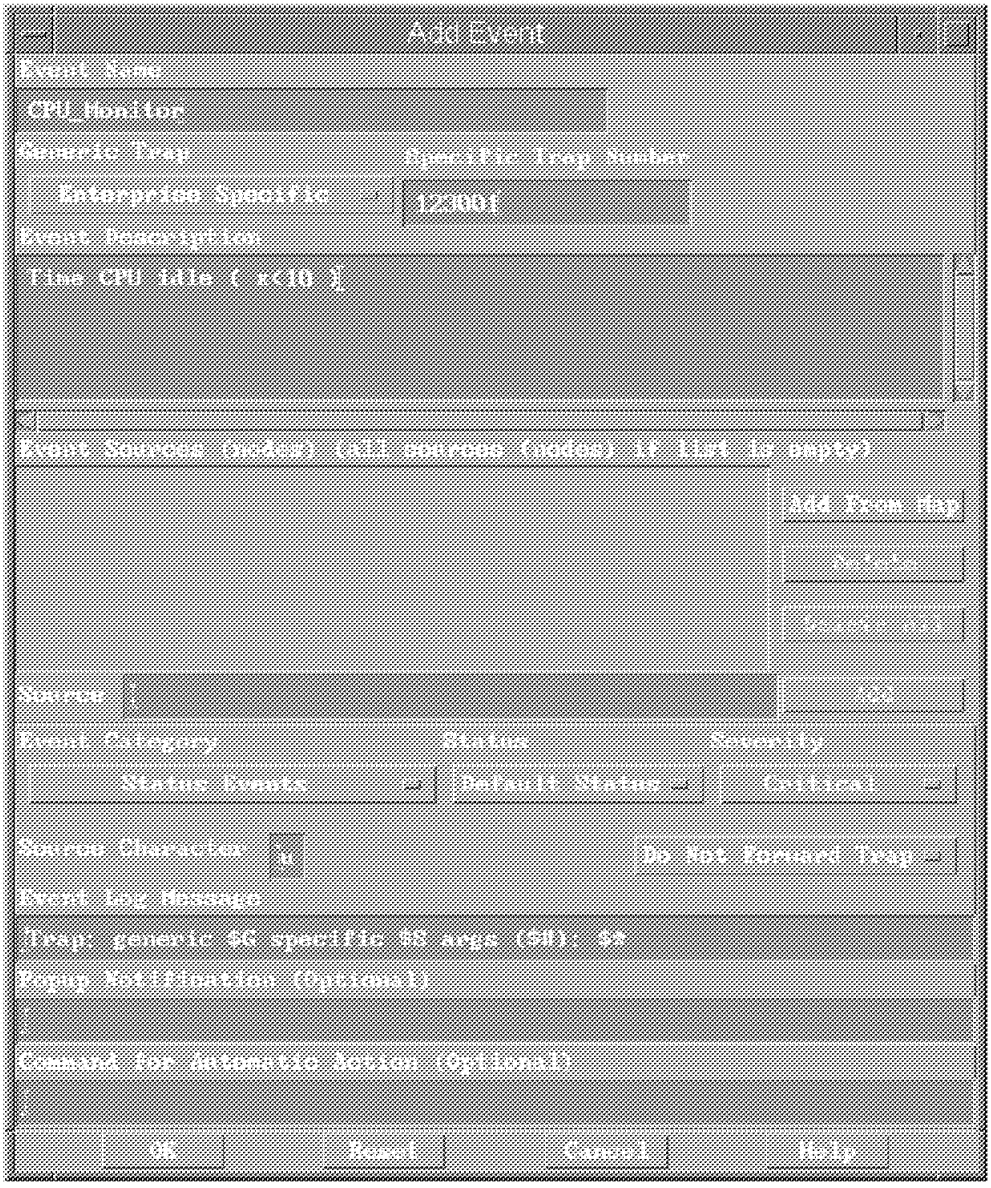


Figure 29. Register Traps -- Configuration of Trap Characteristics

When an event is configured, it is added to the /usr/OV/conf/C/trapd.conf file. The configuration file is used by the SNMP manager to activate the traps received.

Step 3 Set up NetView/6000 for AIX to send the event to T/EC.

On the command line, invoke:

```
smitty nv6000
```

then select **Configure**→**Configure for Tivoli**.

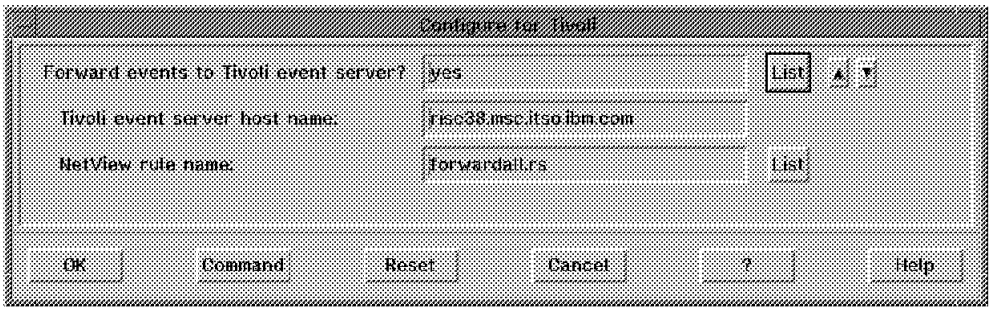


Figure 30. Using SMIT to Configure NetView. Forwarding NetView events to TME 10 Enterprise Console.

Set the Forward Events... option to yes and specify the name of the T/EC server. Select the ruleset name from /usr/OV/conf/rulesets/ entry, then select **forwardall.rs** as the ruleset to use for forwarding all NetView events to the T/EC event console. Note that NetView/6000 for AIX must have the PTF U443133 installed to allow forwarding events to T/EC directly, without having the NetView Adapter installed.

Step 4 To cause these changes to take effect, stop and restart the nvserverd daemon by using the following commands:

```
ovstop nvserverd
ovstart nvserverd
```

To prepare the T/EC server to receive the events, you need to import the event class definitions. There is only one BAROC file containing the single event class used by this adapter: /usr/OV/conf/nvserverd.baroc.

The contents of our sample nvserverd.baroc file is shown in the following example:

```
#!/bin/ksh
###############################################################################
#                                                                             #
#  NetView generic trap                                                       #
#                                                                             #
###############################################################################
TEC_CLASS :
        Nvserverd_Event ISA EVENT
        DEFINES {
                source: default = nvserverd;
                nv_enterprise: STRING;
                nv_generic: INT32;
                nv_specific: INT32;
                nv_var1: STRING;
                nv_var2: STRING;
                nv_var3: STRING;
                nv_var4: STRING;
                nv_var5: STRING;
                nv_var6: STRING;
                nv_var7: STRING;
                nv_var8: STRING;
                nv_var9: STRING;
                nv_var10: STRING;
                nv_var11: STRING;
                nv_var12: STRING;
                nv_var13: STRING;
                nv_var14: STRING;
                nv_var15: STRING;
                };
END
```

The slot defined in the class has the following meaning:

source Set to nvserverd.

nv_enterprise The enterprise ID (MIB object) from the trap. This is the NetView Enterprise ID for internally generated events.

nv_generic The generic trap number.

nv_specific The specific trap number.

nv_varn The values of the variables within the trap or event. Fifteen variables are defined in the BAROC file. Most traps only have a few variables, so having a limit of 15 is not generally a problem.

The adapter also sets the event severity and the event message text (severity and msg are slots inherited from the base event class, so they are not redefined in the nvserverd.baroc file). These values are translated from the values defined when the event is configured in NetView/6000 for AIX. You perform the configuration by selecting **Options→Event Configuration→Trap Customization** from the NetView menu bar. The following section is the step to import the event class into the T/EC server, compile the rule base, load it, and restart the event server.

4.5.3 Setup procedure in T/EC

This section describes the procedure to configure the T/EC event server for receiving NetView events. The steps listed here describe a process similar to the process described in 4.4.1, “How to Forward Events from a Log File” on page 59. However, in these steps we use the equivalent command line approach instead of the TME 10 Desktop.

A rule base contains two types information:

- Event class which defines the structure of an arriving event. The definitions are provided in *.baroc file by the event adapter.
- Event rules which define the actions to be performed when an event is received.

To load the rule base, do the following:

Step 1 Create a rule base in the Event Server Console (rulebase.gif):

```
wcrtrb -d /full/path/name RBname
```

Also refer to the *lwlsrrb* command to find the directory of existing rule bases.

Step 2 Copy the default rule base (class definition and rule) so that it can be imported into the new rule base:

```
wcprb -cr sourceRB destinationRB
```

Step 3 Import a new class file (*.baroc) into the rule base:

```
wimprbclass nvserverd.baroc RBname
```

The import of classes must be done in this order to make sure that the class hierarchy can be resolved.

Step 4 Load and compile the new rule base.

Check the BAROC file class definition:

```
wchkclass nvserverd.baroc RBname
```

Import the BAROC file into rule base:

```
wimprbrules RBname
```

Compile the new rule base:

```
wcomprules RBname
```

Load the new rule base:

```
wloadrb RBname
```

Step 5 Shut down and restart the Event Server.

Shut down the Event server:

```
wstopesvr
```

Restart the Event server:

```
wstartesvr
```

The file /usr/OV/conf/nvserverd.baroc is generated, which can be loaded in the active rule base.

4.5.4 Implement Event Adapter, Event Source, and Group

The *event server* depends on events being received. The basic behavior of the event server is that no events are processed until the event source is defined. *Event adapters* send events to the reception engine of the event server. An *event source* is simply a tag given to an event by the event adapter to assign what type of event adapter sent the event. The value of the event source is defined by the senior administrator. An *event group* is a collection of event filters. An event group is used to logically represent a collection of hosts running a particular application from a given event adapter source.

Create Event Source Select the New event sources menu item from the pop-up menu on the event server icon.

```
wcrtsrc [-S server] [-b bitmap] [-l label] source
```

Event sources must be created by a senior administrator.

Create Event Group Select the New event group menu item from the pop-up menu on the event server icon. Then, create the event group filter.

```
# wcrteg [-s server] [-b bitmap] [event_group]
```

Assign Event Groups Select the Assign Event Group menu item from the pop-up menu on the administrator's event console.

```
# wassigneg console event_group [role ...]
```

4.5.5 Event correlation with NetView/6000 for AIX and T/EC

From PSSP, create a CPU monitor event, then select the SNMP trap options to forward it to NetView, as shown in Figure 31

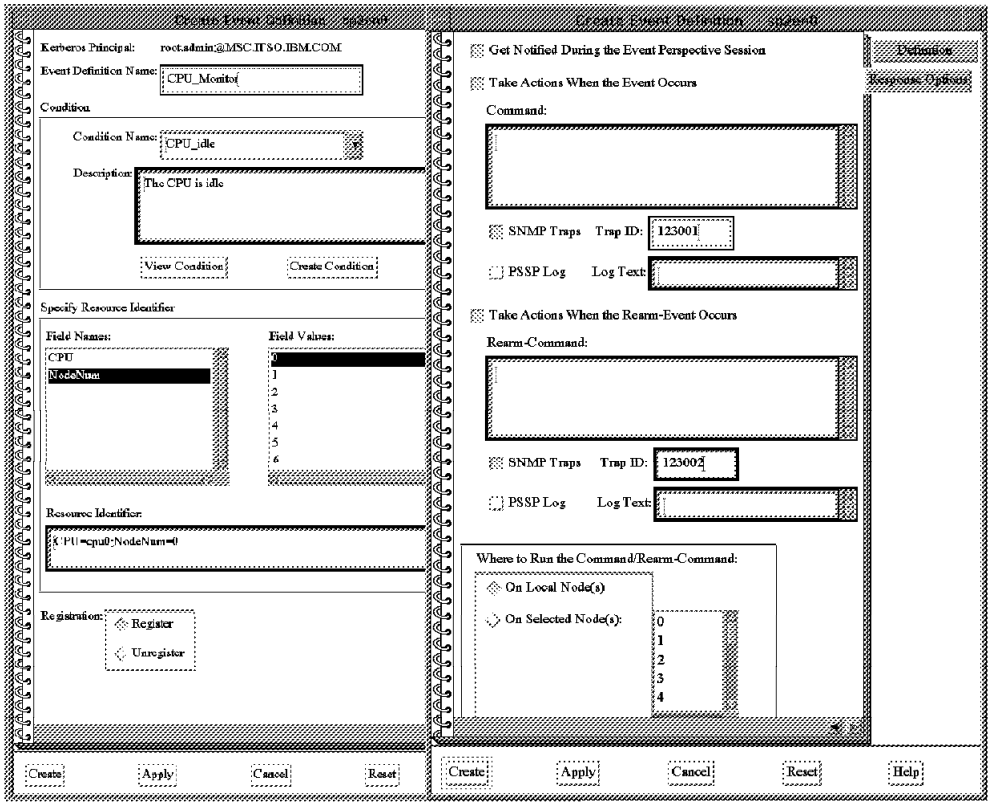


Figure 31. Event Perspectives -- Define an event that generates a trap

The NetView trapd daemon receives this trap and sends it to the event console, as shown in Figure 32 on page 85.

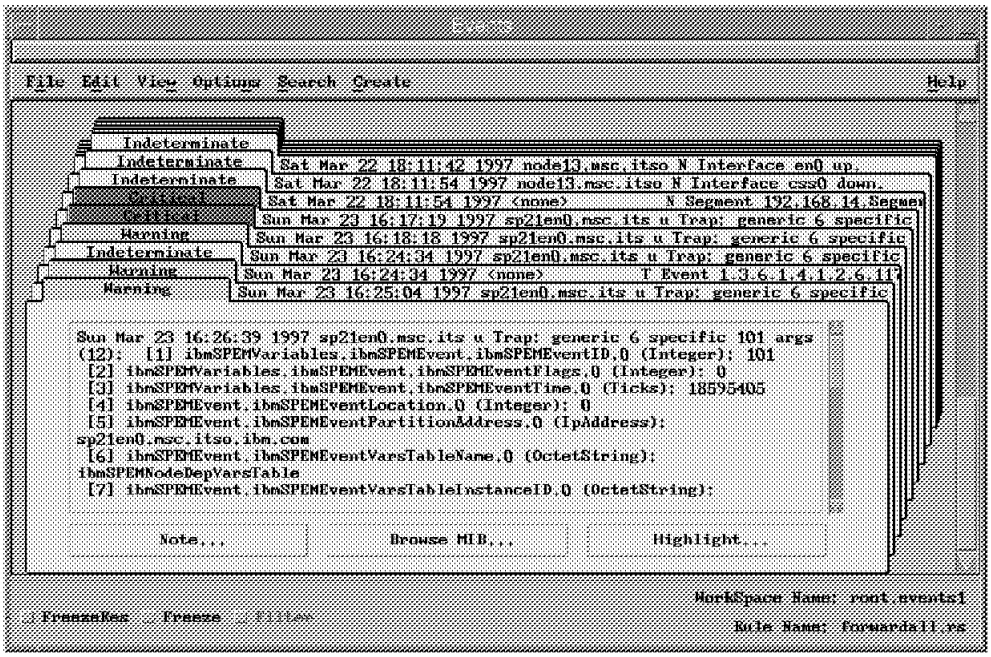


Figure 32. The NetView Event Window -- Receiving SP Traps

The events that are forwarded by the ruleset are then converted into a T/EC event. Figure 33 demonstrates how the events from NetView get presented in the T/EC console.

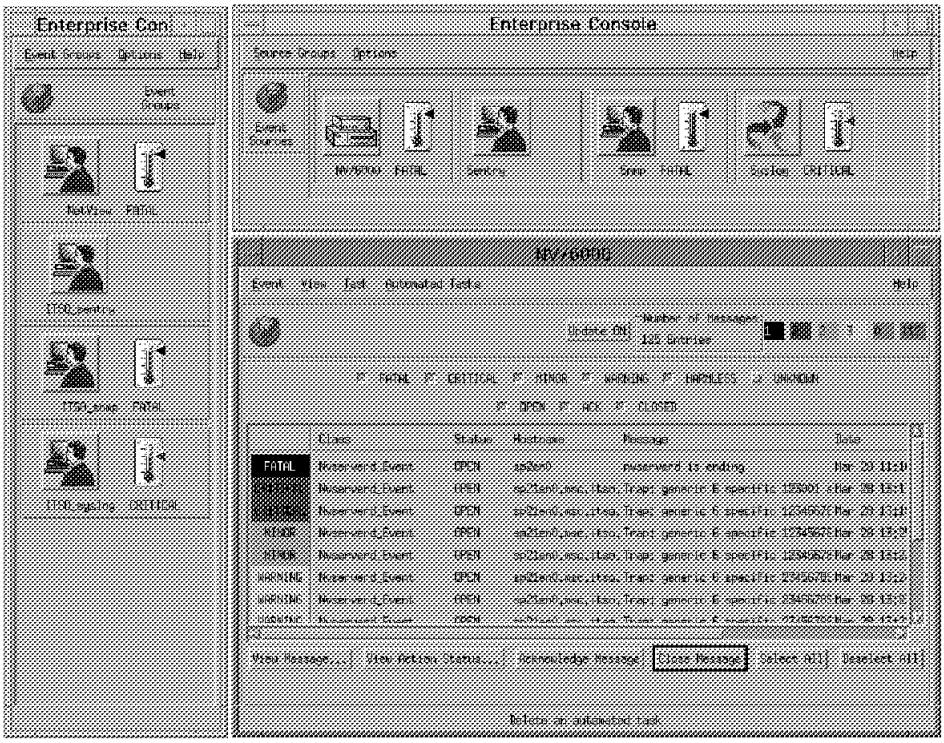


Figure 33. The T/EC Event Console -- Receiving Forwarded Events from NetView

The detailed view, derived from clicking on the event in the T/EC console, shows all detailed information associated with the generated T/EC event. Figure 34 on page 86 shows all the relevant information that was forwarded by NetView and generated by the Problem Management subsystem.

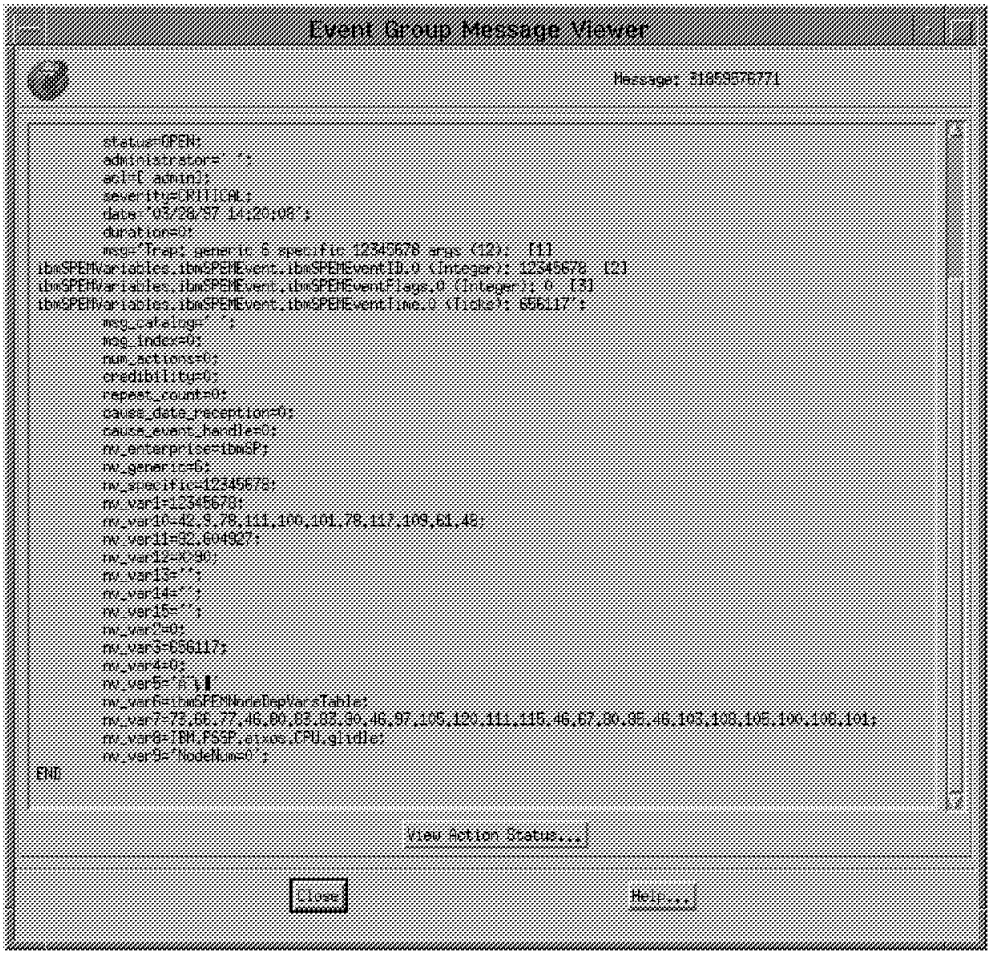


Figure 34. Detailed T/EC View -- Information from NetView Events

4.5.6 Discussion on Event Adapter Implementation Options

There are two ways to get events from NetView/6000 for AIX:

1. By using the NetView V4.1.2 (PTF U443133 installed) built-in ruleset adapter. The NetView/6000 for AIX ruleset adapter uses a NetView event ruleset to decide which events to pass to the T/EC. Furthermore, the event conversion is a generic process, so there is no need to have specific mapping definitions for each event type.

   The ruleset adapter code is embedded in the nvserverd (which is the daemon that passes events to NetView event display). It registers a NetView ruleset to be executed by nvcorrd, the rule processing daemon. Any event that is forwarded by the ruleset is then converted into a T/EC event.

   Note: This is the recommended way to automate forwarding events to T/EC.

2. By using the T/EC NetView Event Adapter. The T/EC NetView Event Adapter uses the OvSNMP API (provided by NetView) to receive raw trap information through registration with the NetView *trapd* daemon, and then generates

T/EC events based on configuration files. This was the previously recommended solution before the TME 10 and SystemView applications were merged.

4.6 Integration of TME 10 Distributed Monitoring and Event Management

Much of the power of recent computer applications lies in the fact that processing is distributed among many server and client machines. Such applications can take advantage of dispersed processing power and the integration of multiple data sources.

From an enterprise system management point of view, however, distributed applications lead to a number of problems. One of these is the problem of monitoring. The system manager needs to be ensured of the health of each component of the distributed system. This can put a strain on the support staff, which has the task of monitoring an ever-increasing number of geographically dispersed systems.

TME 10 Distributed Monitoring provides facilities for monitoring many aspects of a managed system. These include system resources such as CPU utilization, as well as application resources like log files.

TME 10 Distributed Monitoring is composed of two parts: an agent process called the *TME 10 Distributed Monitoring engine*, which performs the monitoring functions on the target system, and a set of monitors. The monitors are the heart of the TME 10 Distributed Monitoring application. They define what resource is to be monitored and what actions are to be taken if a threshold is met or exceeded. These actions normally involve alerting a system administrator to the fact that a threshold has been triggered. However, automated responses can also be executed.

The distribution mechanism for the monitors uses the standard management function of TME 10 Framework for profile subscriptions. For details about the procedures to install and use TME 10 Distributed Monitoring, refer to the ITSO redbook *TME 10 Cookbook for AIX Systems Management and Networking Applications*.

Most of the monitors are predefined for general purpose solutions of system management and are based on the polling mode. The TME 10 Distributed Monitoring engine monitors the managed resources periodically and is always running. It is possible that there needs to be a customized solution to trigger the TME 10 Distributed Monitoring engine to take an action when some situation is detected by the specific application in a push mode.

TME 10 Distributed Monitoring provides the asynchronous facility to fulfill such requirements. Two things are necessary to establish the asynchronous channel: an agent to send an asynchronous message to trigger the TME 10 Distributed Monitoring engine, and the engine configured to wait for the triggering signal and perform some task if the user-defined threshold is met or exceeded.

The asynchronous facility `wasync` allows the agent to send a message to the TME 10 Distributed Monitoring engine on the specified host. The asynchronous message is tagged with the given channel name, data, and optional informational text in the `wasync` command as shown in Figure 35 on page 88.

```
wasync -c channel [-f] [-i info]
       -s data [host]

       -channel    Specifies the name of the asynchronous channel
                   on which transmission takes place.

       -f          Retries the command when the TME 10 Distributed Monitoring
                   engine does not respond, until it succeeds.

       -i info     Specifies text to be sent with the message.
                   The text is passed by TME 10 Distributed Monitoring
                   through any textual notification mechanisms
                   (Tivoli notice, e-mail, file log),
                   and placed in the environment of response
                   programs.

       -s data     Specifies data on which the monitor makes its
                   decisions.

       host        Specifies the name of the host (ManagedNode)
                   whose engine is signalled.
```

Figure 35. Usage of the wasync command

The wasync command will work when the machine from which the command is invoked is a Tivoli managed node.

Two examples are provided in the following sections to illustrate how to combine the asynchronous facility of TME 10 Distributed Monitoring with the PSSP Version 2 Release 2 Problem Management subsystem to perform SP system monitoring tasks.

The first example simply uses the wasync command directly from the Problem Management subsystem to post an event to the TME 10 Distributed Monitoring when a specified threshold is exceeded.

The second example is more complex, but also more robust and flexible. It uses an agent script that receives SP event messages through a named pipe file, analyzes and filters the information, and then uses the wasync command to generate a detailed TME 10 Distributed Monitoring event.

4.6.1 Using the wasync command directly

A simple way to use the TME 10 Distributed Monitoring asynchronous facility is to choose to execute the wasync command directly from the Problem Management subsystem, as shown in Figure 36 on page 89.

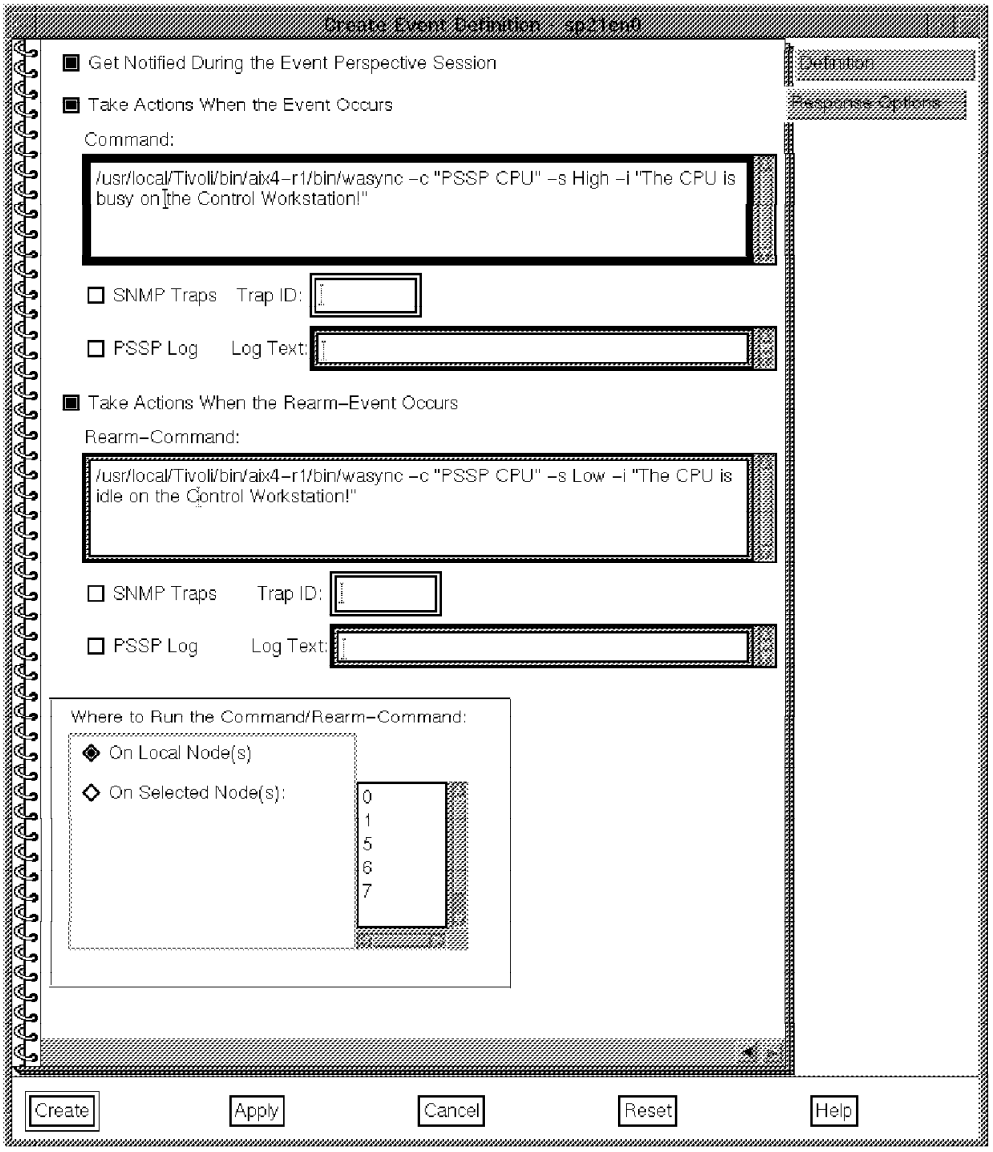


Figure 36. Defining a wasync event using Event Persectives

The equivalent pmandef command would look like the following:

```
pmandef -s SENTRY_PSSP
        -e "IBM.PSSP.aixos.cpu.%idle:CPU=cpu0;NodeNum=0;X<25"
        -c "/usr/local/Tivoli/bin/aix4-r1/bin/wasync -c "PSSP CPU"
            -s High -i "The CPU is busy on the control workstation!"
        -r "X>50"
        -C "/usr/local/Tivoli/bin/aix4-r1/bin/wasync -c "PSSP CPU"
            -s Low -i "The CPU is idle on the control workstation!"
```

When the Problem Management subsystem detects that a threshold has been exceeded, it invokes the wasync command, which sends an event to TME 10 Distributed Monitoring. A pop-up window with the event information will then appear on your screen.

4.6.2 How to integrate the SP Log File with TME 10 Distributed Monitoring

Although this example is more complex than using the wasync command directly from the Problem Management subsystem, it is easily modified to perform SP system management tasks based on the other SP system log files, for example /var/adm/SPlogs/SPdaemon.log. It is a TME 10 Distributed Monitoring alternative to the TME 10 Enterprise Console procedures described in 4.4, "Using the TME 10 T/EC Logfile Adapter" on page 58.

The basic concept of this example is that the PSSP Version 2 Release 2 Problem Management subsystem can be configured to use the system syslog facility to pass an SP-specific message to an agent script through a named pipe file when a user-defined event is generated. As the agent receives the message, it analyzes it and uses the wasync command to trigger the local TME 10 Distributed Monitoring engine immediately. The TME 10 Distributed Monitoring engine uses the data sent by the wasync command to decide which severity level threshold is met or exceeded, and takes a user-defined action.

Steps one to seven prepare the parsing method for TME 10 Distributed Monitoring to understand the type of events, generated in the log file.

Step 1 On the control workstation, alter the configuration file of the system syslog daemon to get SP-specific messages generated by the PSSP Problem Management subsystem:

```
cat >> /etc/syslog.conf <<EOF
daemon.notice       /tmp/log_pssp.log
EOF
```

Step 2 Create the log file that will store the SP-specific messages:

```
/usr/bin/touch /tmp/log_pssp.log
```

Step 3 Notify the system syslog daemon that the configuration file has been changed:

```
/usr/bin/refresh -s syslogd
```

Step 4 Use the Event Perspectives GUI or the pmandef command to define an event to monitor the CPU utilization on control workstation sp21en0 with the predefined condition, idlecpu. The event definitions are shown in Figure 37 on page 91. (For a description of the procedure to define such an event, refer to *IBM PSSP for AIX Administration Guide*.

When CPU idle time is less than twenty-five percent, Problem Management writes a PSSP log entry to the syslog file to indicate a CPU busy situation. When CPU idle time is greater than fifty percent, Problem Management will write a PSSP log entry to the syslog file to indicate a CPU idle situation.

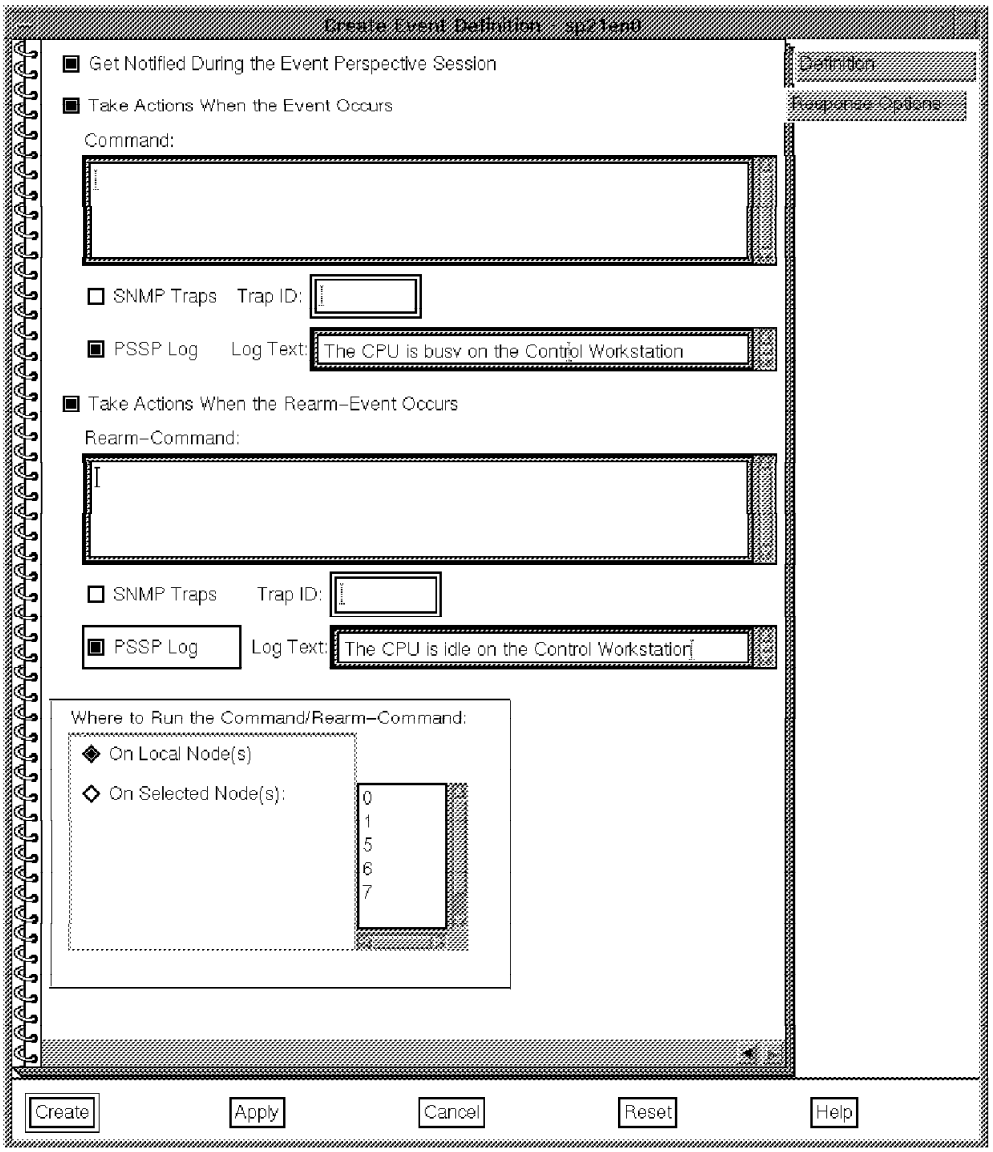


Figure 37. Defining a Log File Event Using Event Perspectives

The equivalent pmandef command looks like the following:

```
pmandef -s SENTRY_PSSP
        -e "IBM.PSSP.aixos.cpu.&idle:CPU=cpu0;NodeNum=0:X<25"
        -l "The CPU is busy on the control workstation!"
        -r "X>50"
        -L "The CPU is idle on the control workstation!"
```

Step 5 Analyze the string structure of the following two major SP-specific messages written by Problem Management in the log file /tmp/log_pssp.log. The purpose of this step is to match each of these two SP-specific log file entries to a unique mapping pattern. The following message patterns are found in the /tmp/log_pssp.log file:

- When the CPU is busy:

```
Mar 20 14:31:23 sp21en0 pmand[39444]: SP Problem Mgmt:
Monitored Situation Exists:
Name=LOGFILE_PSSP_CPU
Node Number=0
Resource Variable=IBM.PSSP.aixos.cpu.idle
Instance Vector=CPU=cpu0;NodeNum=0
Predicate=X<25
Description=The CPU is busy on the control workstation!
```

- The following entry is found in the /tmp/log_pssp.log file when the CPU is idle:

```
Mar 20 14:32:08 sp21en0 pmand[39444]: SP Problem Mgmt:
Monitored Situation Cleared:
Name=LOGFILE_PSSP_CPU
Node Number=0
Resource Variable=IBM.PSSP.aixos.cpu.idle
Instance Vector=CPU=cpu0;NodeNum=0
Predicate=X>50
Description=The CPU is idle on the control workstation!
```

Step 6 Design and code the agent script according to the results from Step 5:

```
# cat > log_pssp.ksh <<EOF
#!/bin/ksh

# Open the named pipe as standard input
exec < /tmp/log_pssp.fifo
while read line; do
  case "$line" in
       *pmand*LOGFILE_PSSP_CPU*Predicate=X>50*(
              /usr/local/Tivoli/bin/aix4-r1/bin/wasync
                     -c "PSSP CPU"
                     -s Low
                     -i "The CPU is idle on control workstation!"
              logger "The LOGFILE_PSSP_CPU_Low has been processed!"

       *pmand*LOGFILE_PSSP_CPU*Predicate=X<25*(
              /usr/local/Tivoli/bin/aix4-r1/bin/wasync \
                     -c "PSSP CPU"
                     -s Low
                     -i "The CPU is busy on control workstation!"
              logger "The LOGFILE_PSSP_CPU_High has been processed!"

       *(

  esac
done
EOF
#
```

Step 7 Create a TME 10 Distributed Monitoring profile and a subscription of type asynchronous string that looks for messages from channel PSSP CPU:

- Open a profile manager, LOGFILE_Sentry, and create the TME 10 Distributed Monitoring profile, LOGFILE_SENTRY.
- Open the profile and click **Add Monitor**.
- Select **Asynchronous** (**string**) (Monitoring Sources) from the choice list of Unix_Sentry or Sentry 2.0 (Monitoring Collections) and give the channel name (PSSP CPU), then click **Add Empty...**.
- In the Edit Sentry Monitor dialog, select **Matches** for the critical response level, and enter High in the text field. Choose the **Pop-Up** response action.

- In the Edit Sentry Monitor dialog, select the relation **Matches** for the warning response level, and type Low in the text field. Choose the **Pop-Up** response action.
- Click **Change & Close**. Note that there is no **Schedule** button because this is an asynchronous criterion.
- Save the profile and distribute it to the subscriber, which is the control workstation.

The following steps install and define the operating environment for TME 10 Distributed Monitoring, to start monitoring the log file entries.

Step 8 Remove the entry added in Step 1 from the configuration file of the system syslog daemon, /etc/syslog.conf:

```
vi /etc/syslog.conf
```

Step 9 Alter the configuration file of the system syslog daemon to get SP-specific messages generated by the PSSP Problem Management subsystem.

```
cat >> /etc/syslog.conf <<EOF
daemon.notice       /tmp/log_pssp.fifo
EOF
```

Step 10 Create the named pipe file that will receive the SP-specific messages:

```
/etc/mknod /tmp/log_pssp.log p
```

Step 11 Run the agent script in the background to receive SP-specific messages from the predefined named pipe file:

```
./log_pssp.ksh &
```

Step 12 Notify the system syslog daemon that the configuration file has been changed:

```
/usr/bin/refresh -s syslogd
```

> **Note:**
>
> Make sure that the agent script is running before executing this step, or the syslogd daemon may be hung by writing a log entry to a pipe file that does not have a reading process.

Step 13 Start up the administrator's TME 10 Desktop:

```
. /etc/Tivoli/setup_env.sh
tivoli
```

When the CPU thresholds are exceeded, one of the two pop-up windows shown in Figure 38 on page 94 will appear on your screen to reflect the CPU utilization on the control workstation.

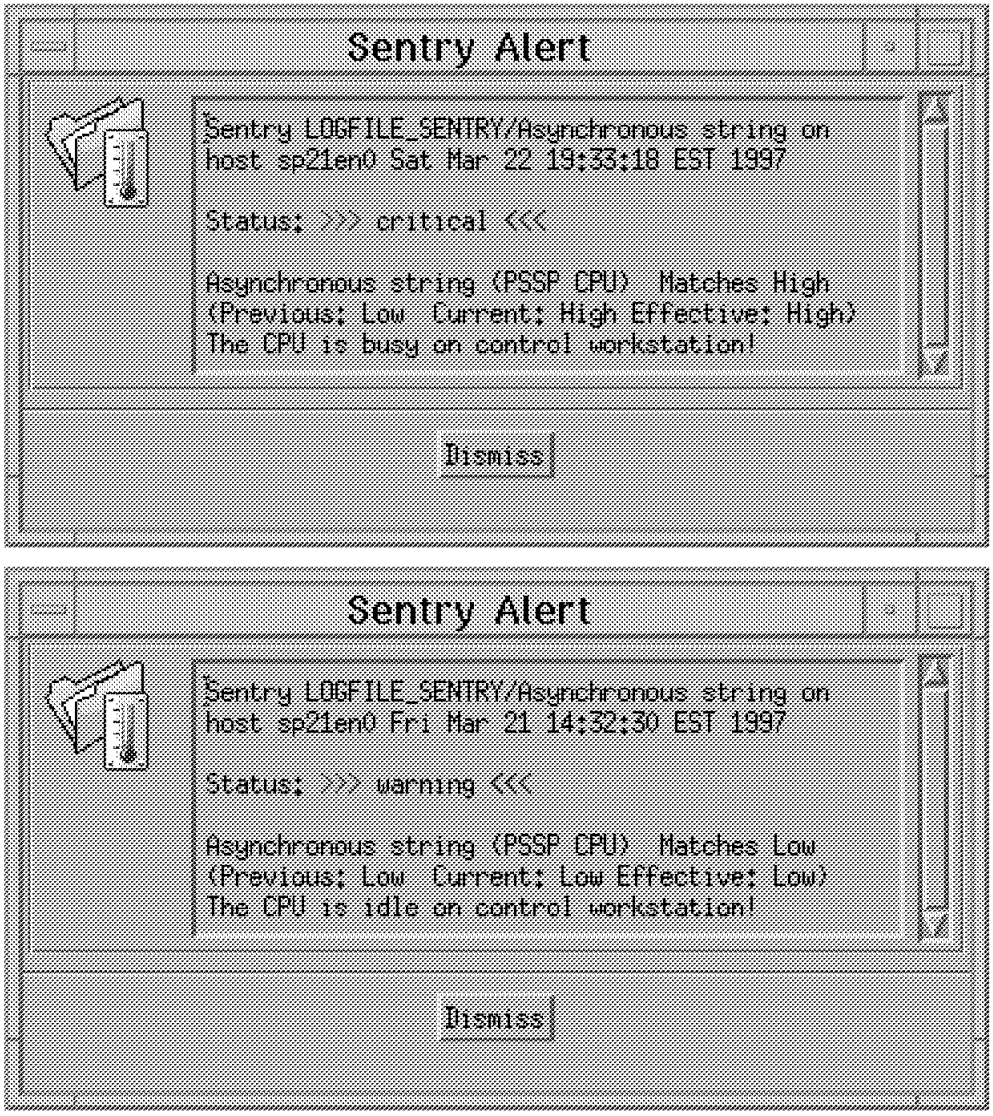


Figure 38. TME 10 Distributed Monitoring Pop-Up for CPU idle

Chapter 5. Task Libraries, Tasks, and Jobs

The TME 10 Framework task libraries can be used to provide a simple interface to executing common RS/6000 SP commands. A *task library* is a collection of tasks that have been defined to the system. A *task* is simply a command or script that Tivoli can execute on behalf of the administrator. Execution roles and userids can be assigned to a task, allowing you to control who can execute the task and how that task is to be executed.

To support your RS/6000 SP from Tivoli, tasks can be defined to execute local scripts that invoke a specific SP command or a set of commands to perform routine SP administrative operations. A task can query system data, power nodes on or off, launch Perspectives and other applications, manage the switch, or perform just about any function that is routinely executed by an SP administrator.

Once you have created a set of tasks in a task library, you can bind an instance of one of those tasks to a set of managed nodes or profile managers to create a *job*. Jobs contain execution-specific information and can be copied directly to an administrator's TME 10 Desktop. This allows another level of control over which administrators are authorized to perform specific SP tasks.

5.1 General Procedure for Creating Tasks

The simplest way to create a task is to use the TME 10 Desktop. The GUI allows you to create simple tasks that do not require input parameters. If you have a task that does require additional input to execute, you must develop the task using the Task Library Language. In addition to the information provided in the *TME 10 Task Library Language Developer's Guide*, the *TME 10 Cookbook for AIX Systems Management and Networking Applications* provides excellent detailed descriptions on the process of creating task libraries, tasks and jobs. The *Cookbook* also contains a very thorough explanation of using the Task Library Language and implementing tasks requiring input as a way of adding function to the TME 10 Desktop. It is highly recommended that the information in that reference is understood before proceeding with the process outlined in this chapter.

The general process of creating a task using the TME 10 Desktop is as follows:

1. Create a policy region to contain your task libraries, or use an existing policy region.

   Since many SP commands require root authority to execute, it is recommended that you create a separate policy region for SP task libraries. You will need to modify the Managed Resource Policies for the TaskLibrary resource such that the Validation Policy allows setting the root userid for task execution.

2. Within the policy region, create a task library.

3. Within the task library, create a task specifying the script to execute.

   Note: When creating a task, Tivoli will make a complete copy of the executable that you specify when defining the task. If you specify a binary executable, not only will making this copy use unnecessary disk space, it will slow down execution time since Tivoli ships the executable to the target

machine to be executed. Also, since Tivoli maintains a copy of the executable at the time the task was defined, you will need to delete and recreate the task any time the executable is updated. Therefore, we recommend that you write simple scripts as wrappers to the SP commands that you wish to execute, and do not specify the SP command directly when defining a task.

5.2 SP Task Libraries

Creating tasks to execute a script on the SP is a simple process. You can build your own set of tasks tailored to the SP work your administrators perform most often. This redbook includes two simple task libraries that you can use as examples for creating your own tasks.

The first task library, called SPTasks, includes general tasks to perform the following:

- List data for the SP using the `splstdata` command.
- List data for the SP node the task is run on using the `splstdata` command.
- Run any specified command.
- Launch the Perspectives application as a separate process.
- Launch a specified application as a separate process. This might be useful for starting applications such as SMIT on your SP control workstation.
- Run the `cstartup` command on the control workstation using specified parameters. This can be used to power on one or more nodes.
- Run the `cshutdown` command on the control workstation using specified parameters. This can be used to power off one or more nodes.

The second task library, called SwitchTasks, includes tasks that are specific to managing your SP switch:

- List data for the SP switch using the `splstdata` command.
- List the switch responds information using the `SDRGetObjects` command.
- List the primary node information returned by the `Eprimary` command.
- Set the specified oncoming primary and backup nodes using the `Eprimary` command.
- Quiesce the switch for the specified partition.
- Start the switch for the specified partition.
- Fence the specified nodes.
- Unfence the specified nodes.

The source listings for these task libraries are included in Appendix G, “Sample Task Library Listings” on page 193. They are also included with the software distributed with this redbook. You may use these listings as examples to create your own tasks. To load the task libraries and use them directly in your environment, see 5.3, “Using the Task Library Language” on page 97.

There are many other tasks that you may wish to create. For example, if your administrators are often invoking SMIT to do their work, you may wish to write a task to invoke SMIT directly, similar to the one that invokes Perspectives, instead

of using the task that launches any application. If you often check to see which daemons are active, you can write a task to invoke the `lssrc -a` command. The possibilities here are endless.

5.3 Using the Task Library Language

The Task Library Language is described in the *TME 10 Task Library Language Developer's Guide*. While this guide provides a description of the language, it is difficult to understand the process for creating and loading your own tasks. Refer to the *TME 10 Cookbook for AIX Systems Management and Networking Applications* for an excellent review of this process. We will only provide a high-level overview of the process here.

You can load the task libraries distributed with our redbook by using the following steps:

1. Load the SPTasks.tll and SwitchTasks.tll files from the diskette onto a local filesystem.
2. Import the task libraries into Tivoli by using the following commands from that local filesystem directory:

   ```
   wtll -p PolicyRegion -P /bin/cat SPTasks.tll
   wtll -p PolicyRegion -P /bin/cat SwitchTasks.tll
   ```

   where PolicyRegion is the name of the existing policy region where you want to place the task libraries.

You can then start the TME 10 Desktop and open your policy region. You should see two new task libraries similar to those shown in Figure 39 on page 98.

Figure 39. SP Task Libraries. Two task libraries, SPTasks and SwitchTasks, are distributed with this redbook.

If you open the two new task libraries, you will see the provided tasks as shown in Figure 40 on page 99 and Figure 41 on page 99.

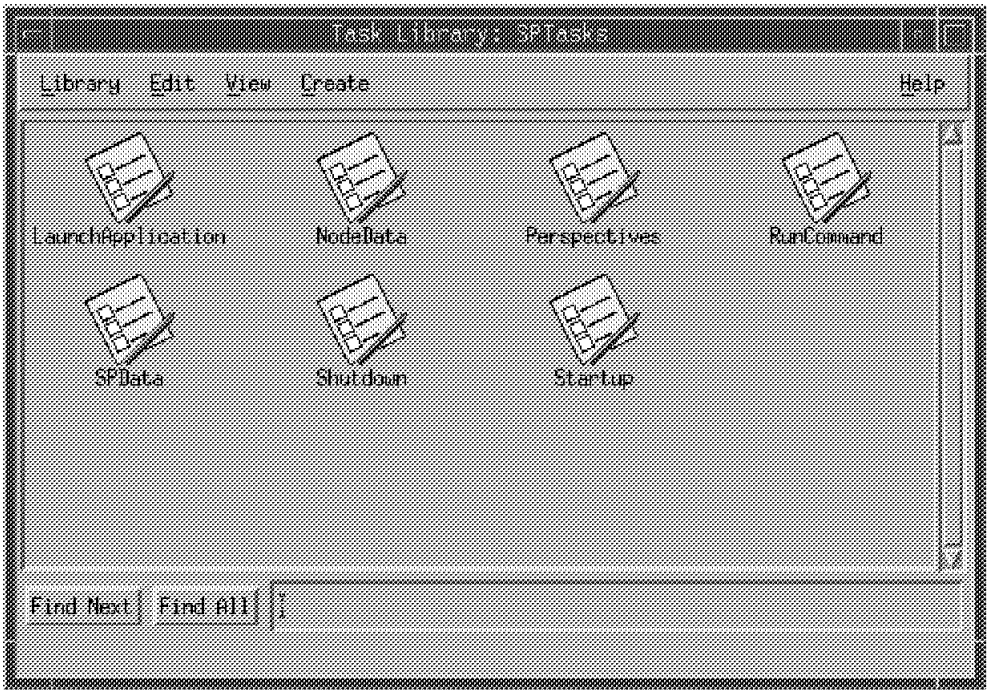


Figure 40. Tasks in the SPTasks Library. These tasks are examples of some general SP tasks you may wish to invoke from Tivoli.

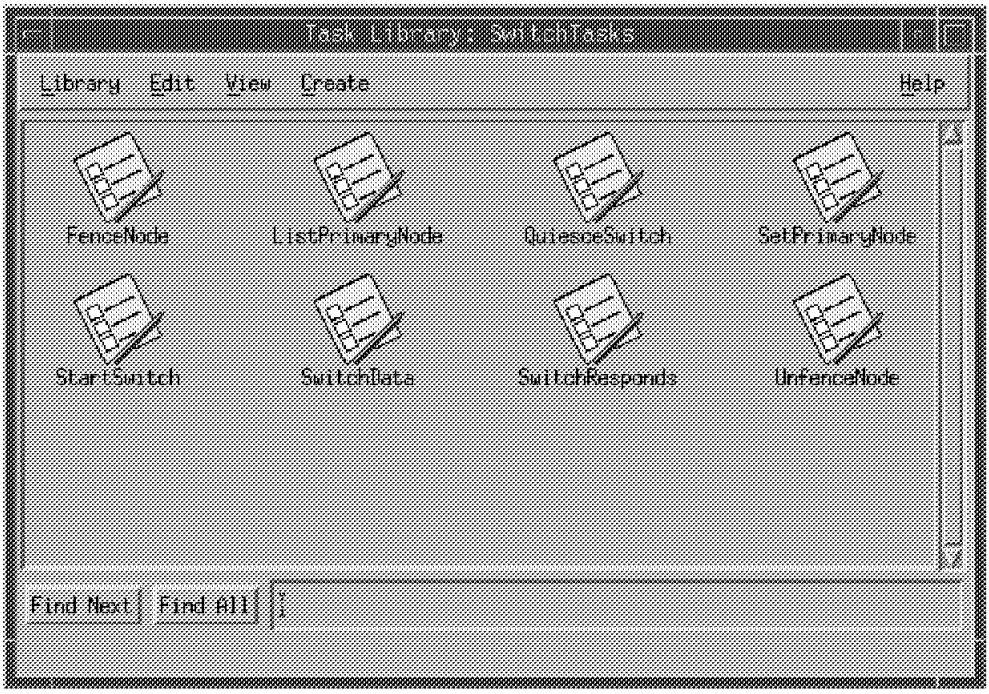


Figure 41. Tasks in the SwitchTask Library. These tasks are examples of some common SP switch tasks that you may wish to invoke from Tivoli.

5.3.1 Creating Customized Tasks

There are two ways to create your own customized tasks:

- You can use the task libraries distributed with this redbook as a starting point, and modify them to include your own customized tasks.
- You can create your own new task library with only the tasks you need.

To customize the distributed task libraries, do the following:

1. Install the task libraries following the directions in 5.3, “Using the Task Library Language” on page 97.
2. Edit the SPTasks.tll and SwitchTasks.tll files and add your own tasks. You can use the existing tasks in those files as examples. Also, review the chapter on “Adding Function to the Tivoli Desktop.” in the *TME 10 Cookbook for AIX Systems Management and Networking Applications* to help with this step.

   You can create a new task library by copying one of those tll files to a new file, changing the name of the task library, and replacing the existing tasks with your own.
3. After your source file is updated, import the task library into Tivoli. If you are replacing an existing task library, use the following command:

   ```
   wtll -r -p PolicyRegion -P /bin/cat tll
   ```

   If you have changed the name of the task library and are creating a new one, omit the **-r** option.

   If you get errors while loading your task library, look carefully at the syntax of your source file. The diagnostic messages are not very helpful. See the *TME 10 Cookbook for AIX Systems Management and Networking Applications* for some pointers.
4. Test your new tasks by trying to execute them. If you find problems in your executable scripts, you will need to edit the original tll file and reload the task library.

The easiest way to create a new task library with the Task Library Language is to export an existing task library from the TME 10 Framework and modify it:

1. Using the TME 10 Desktop, create a new task library in your policy region.
2. Create a simple task in that task library.
3. From the AIX command line, export the task library using the command:

   ```
   wtll -F TargetFile -l TaskLibrary
   ```

   where TargetFile is the file name of a tar file that will be created containing the exported task library, and TaskLibrary is the name of the task library to be exported.
4. Extract the files from the exported TargetFile:

   ```
   tar -xvf TargetFile
   ```
5. Continue with the previous steps to modify an existing task library and reload it into your framework.

You must use the Task Library Language to create a task that requires input from the user. The task libraries distributed with this redbook and listed in

Appendix G, “Sample Task Library Listings” on page 193 contain several examples of these types of tasks.

For example, to execute the RunCommand task, the user must specify the command to run. The listing for this task shows how to specify simple text input. Here is an excerpt of this task:

```
TaskLibrary "SPTasks" {

        ...

        ArgLayout inputText {
                Text;
        };

        ...

        Task RunCommand {
            Description = ("_!_","Execute any command",1);
            HelpMessage = ("_!_","Input the command and any parameters",1);
            Uid = ("_!_","*",1);
            Comments = ("_!_"," ",1);
            Roles = ("_!_","admin",1);
            Argument ("_!_","Command to run: ",1) {
                    Layout = ("","inputText",1);
            };
            Implementation ("default")
.#!/bin/ksh
.export PATH=/usr/lpp/ssp/bin:/tivoli/bin/aix4-r1/CUSTOM:$PATH
. . /etc/Tivoli/setup_env.sh
.export DISPLAY=$WD_DISPLAY
.$1
            ;
        };

        ...
```

The Argument and ArgLayout constructs are one way to specify input text parameters.

Another good example to look at, as shown in the following screen, is the StartSwitch task in the SwitchTasks library. This shows an example of how you can create a choice selection list of all the SP partitions and then how to execute an SP command for the partition selected by the user.

```
TaskLibrary "SwitchTasks" {

        ...

        ArgLayout Partitions {
                TextChoice Program {
                   Implementation ( "default" )
.#!/bin/ksh
.for i in /usr/lpp/ssp/bin/splst_syspars -n
.do
.  echo $i
.done
       ;
 };
 };

 ...

   Task StartSwitch {
       Description = ("_!_","Start the switch in the specified partition",1);
       HelpMessage = ("_!_","Select partition to start switch in",1);
       Uid = ("_!_","root",1);
       Comments = ("_!_"," ",1);
       Roles = ("_!_","admin",1);
       Argument ("_!_","Partition to start switch in: ",1) {
               Layout = ("","Partitions",1);
       };
       Implementation ("default")
.#!/bin/ksh
.estart="/usr/lpp/ssp/bin/Estart"
.if [[ ! -a $estart ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.# set the current partition
.syspar=$1
.$estart
           ;
        };

        ...
```

In this case, ArgLayout specifies a program for the TextChoice, and the implementation is imbedded directly in the task library source.

When you create a job from a task that requires input, you can bind the task to a specific set of target managed nodes. For SP commands, this will usually be the control workstation. However, you cannot specify the input parameters until the job is actually run. You may find yourself creating many small, special-purpose tasks for those commands that do require input so that you can run them as scheduled jobs or in response to messages received by your TME 10 Enterprise Console.

One last example that is useful to look at is the task to invoke Perspectives as a separate process, as shown in the following screen:

```
TaskLibrary "SPTasks" {

        ...

        Task Perspectives {
            Description = ("_!_","Invoke Perspectives as a separate process",1);
            HelpMessage = ("_!_","No Help Available",1);
            Uid = ("_!_","*",1);
            Comments = ("_!_"," ",1);
            Roles = ("_!_","admin",1);
            Implementation ("default")
.#!/bin/ksh
.persp="/usr/lpp/ssp/bin/perspectives"
.if [[ ! -a $persp ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH
. . /etc/Tivoli/setup_env.sh
.export DISPLAY=$WD_DISPLAY
.# We close stdin, stdout and stderr
.exec 0<&-
.exec 1<&-
.exec 2<&-
.$persp &
            ;
        };

        ...
```

Notice that in the implementation section, we close the stdin, stdout, and stderr file descriptors before invoking Perspectives as a background process. If we did not close these file descriptors first, the TME 10 Desktop would not return control until the process completed (in this case, you exited Perspectives).

Chapter 6. AEF Customizations for the RS/6000 SP

Tivoli's current technology permits the management of RS/6000 SP nodes as a set of AIX workstations. User and group management, host management, software distribution, job scheduling and other basic system management tasks can be effectively implemented for the SP without the need of any modifications to the TME 10.

When Tivoli's client software has been loaded on SP nodes and the control workstation, these appear in the TME 10 Desktop as AIX managed nodes. Hence, the standard informational dialogs (windows) and capabilities of any managed node (like subscribing to profile managers) are available.

However, system management capabilities specific to the SP, like hardware monitoring, are not available in a standard Tivoli installation. In order to capture the specific characteristics of the RS/6000 SP hardware and software, we have customized the managed node resource to behave as either an SP node or as an SP control workstation.

With these customizations, managed nodes appear in the TME 10 Desktop as SP-specific objects, with extended dialogs and menus. Their standard dialogs are not overridden, but merely extended. Hence, customized nodes are treated as AIX managed nodes for all Tivoli standard system management functions.

With our customizations, the SP control workstation object has extended dialogs to:

- List (and modify when the administrator has the appropriate permissions) its SDR data.
- List all SP nodes it controls.
- Run commands against any number of nodes (regardless of whether or not they are managed nodes).
- Power on/off any set of nodes.
- Fence and unfence any set of nodes.
- Launch SP applications, like Perspectives or the System Monitor GUI.

The SP node object has new dialogs to:

- List (and modify with the appropriate permissions) the SDR data.
- Run commands against the node.

These customizations are easy to modify to suit your system management environment's needs. Furthermore, the user is at liberty to install these customizations in any number of SP nodes or SP control workstations that are managed nodes in a Tivoli environment. The customization of a managed node as either an SP node or an SP control workstation is self-contained and is not affected by the customization of other nodes or control workstations. For example, you can install the customizations only for the SP control workstation and you should still be able to power on/off nodes and perform the other management functions provided for the control workstation object, even when the SP nodes are not customized as such, or even when they have not been defined as Tivoli managed nodes.

We have used the Application Extension Facility (AEF) to perform these customizations. AEF is a Tivoli application toolkit that allows for the easy extension of the behavior of the TME 10 Desktop resources. This chapter contains a basic introduction to AEF development. The code for these customizations, as well as installation and uninstallation scripts are listed in Appendix H, “Contents of AEF Customization Scripts” on page 201 and provided on the attached diskette. This code is free, provided on an “as is” basis.

The main goal of this chapter is to discuss this sample set of customizations: how they were made, how to install them, and how you can easily modify them or add your own customizations. We believe that the integration they provide with the TME 10 Desktop will be very beneficial to customers who wish to fully manage the SP as part of their TME.

Since the main topic of this chapter is the integration of the SP system management functions and the TME 10 Desktop, basic familiarity with both is assumed. For more information on these topics consult the TME 10 manuals, as well as the *IBM PSSP for AIX Administration Guide*.

The material in this chapter is organized as follows:

- 6.1, “High Level Overview of a Sample Set of Customizations” on page 107.

  This section provides a high-level overview of a sample set of AEF customizations that extend managed nodes as either SP nodes or as the SP control workstation.

- 6.2, “Installation of the Sample Customizations” on page 113.

  This section provides the installation procedures for the previously-mentioned set of customizations.

Users that are interested in understanding how these customizations were made and in how to produce their own, should review the rest of the chapter, which provides an overview of the TME 10 Application Extension Facility.

- 6.3, “What is AEF?” on page 115.

  This section provides an introduction to AEF, followed by a discussion of its strengths and weaknesses.

- 6.4, “Anatomy of an AEF/DSL Customization” on page 119.

  This section provides a description of the basic steps of installing and uninstalling customizations.

- 6.5, “TME 10 Desktop Dialogs” on page 123.

  This section provides instructions on how to modify existing dialogs, or how to design your own, using the Dialog Specification Language (DSL).

- 6.6, “Methods for Customized Objects” on page 129.

  Every gadget in a dialog is associated with a method, which can be user defined. This section provides a description of how to implement your own methods, or how to use those provided by TME 10 method libraries.

- 6.7, “Bitmaps” on page 133.

  Every TME 10 resource has a set of bitmaps associated with it. This section provides an explanation of how these bitmaps can be customized.

- 6.7.1, “Icons” on page 134.

Icons are the graphical representations of resources in the TME 10 Desktop. Formally, an *icon* is defined as a bitmap and its associated pop-up menu. This section provides a description of how to customize icons to specifically represent customized objects.

- 6.8, “Messages and Message Catalogs” on page 136.

  AEF/DSL provides a basic internationalization support through message catalogs. We have used such message catalogs extensively in our sample customizations. This section discusses how to modify the existing message catalogs, or implement new ones.

The present chapter is complemented by Appendix H, “Contents of AEF Customization Scripts” on page 201, where the dialogs and callbacks of our sample customization are listed. The complete source code for the customizations is provided on the diskette distributed with this redbook.

6.1 High Level Overview of a Sample Set of Customizations

This section provides a functional overview of the sample customizations of Tivoli managed nodes that we have implemented to accommodate SP system management within the TME 10 Desktop. The next sections discuss them in detail, through examples that users can easily modify to come up with their own customizations.

The target user for these customizations is a Tivoli user that utilizes TME 10 to manage an installation that includes RS/6000 SP systems. We assume this user wants to execute routine administration functions specific to the RS/6000 SP, without abandoning the Tivoli environment.

With this goal in mind, we have designed the customizations to *increment* the current functionality of the Tivoli environment, not to override it. Hence, the managed node has been enhanced with new dialogs, callbacks, icons and message catalogs facilities, to provide basic internationalization support.

We have extended the managed node behavior to define two new objects: sp_node and sp_cws. These objects appear with SP-specific icons on the TME 10 Desktop (as in Figure 42 and Figure 43 on page 108). When their **properties** pop-up menu option has been selected, an extended dialog appears to provide not only standard information about the OS Version or Hostname, but also SP specific data and functionality.

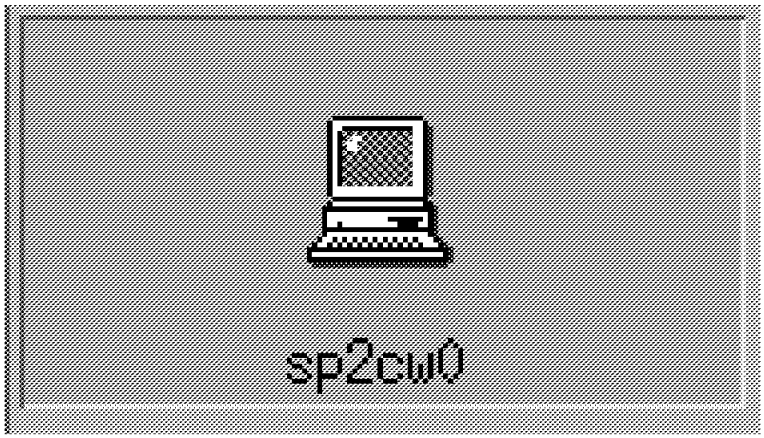


Figure 42. A Managed Node customized as an SP Control Workstation

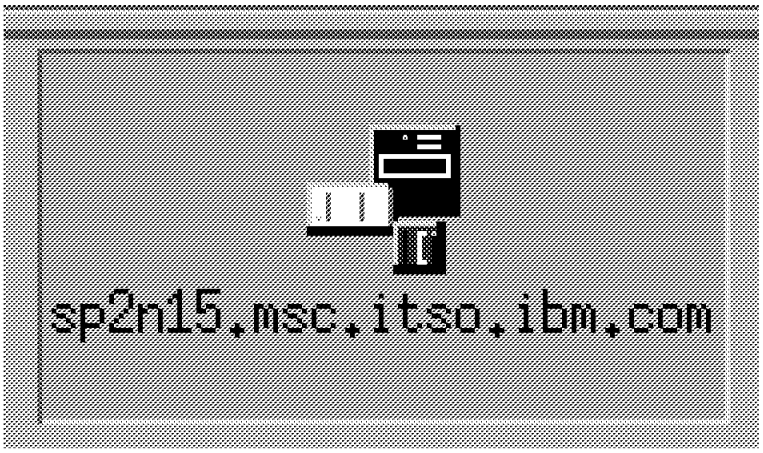


Figure 43. A Managed Node customized as an SP Node

Specifically, when a managed node has been defined as an sp_node, its properties dialog appears with extended capabilities (as indicated in the SP Properties frame in Figure 44 on page 109).

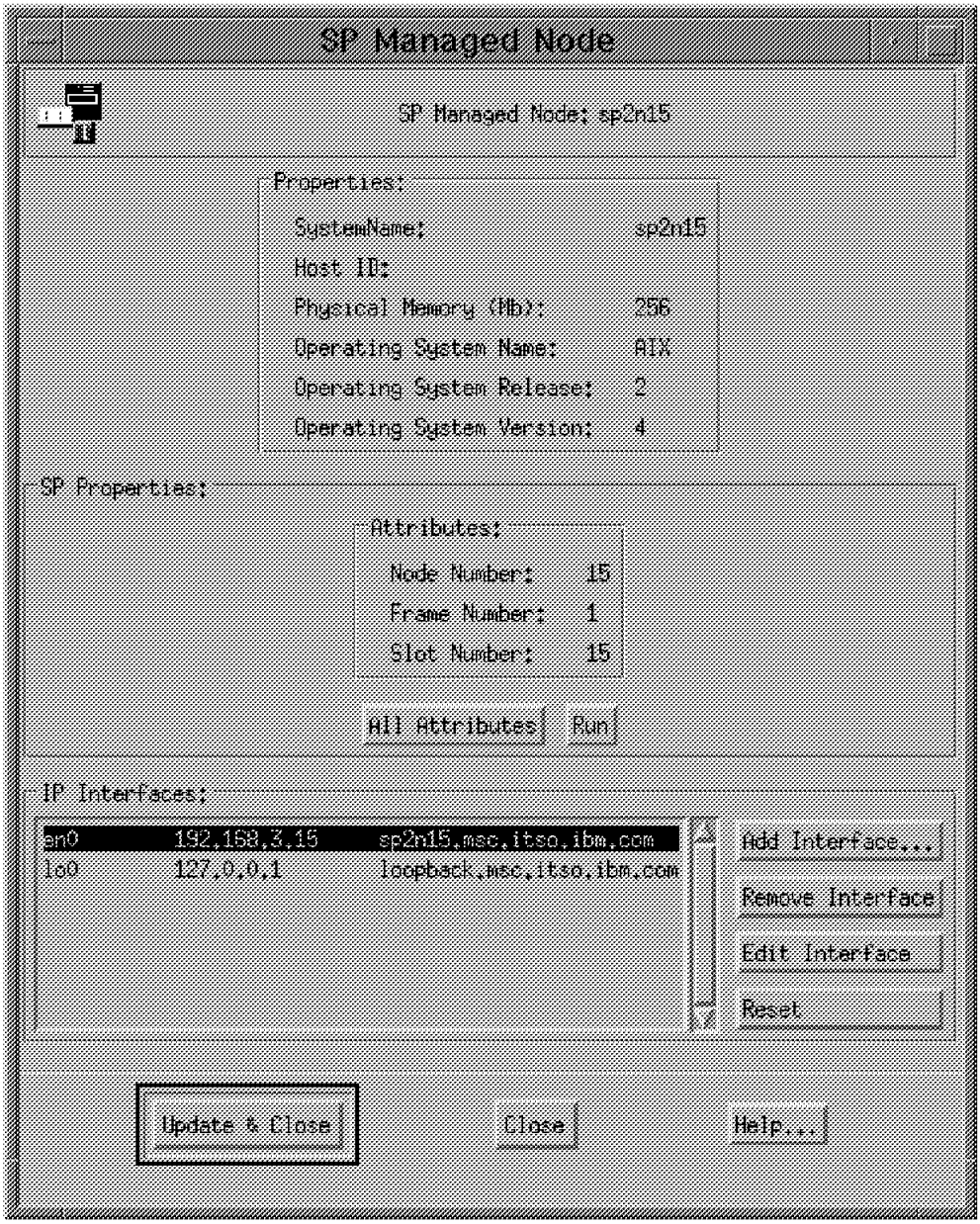


Figure 44. Extended Managed Node Properties

In addition to the standard functionality of the properties dialog, the following capabilities are provided for the sp_node object:

1. System Repository Data associated with the node can be displayed and changed (provided the administrator has the authorization role of super). A new dialog displaying the SDR attributes appears when the user clicks on the **All Attributes** button, as shown in Figure 45 on page 110.

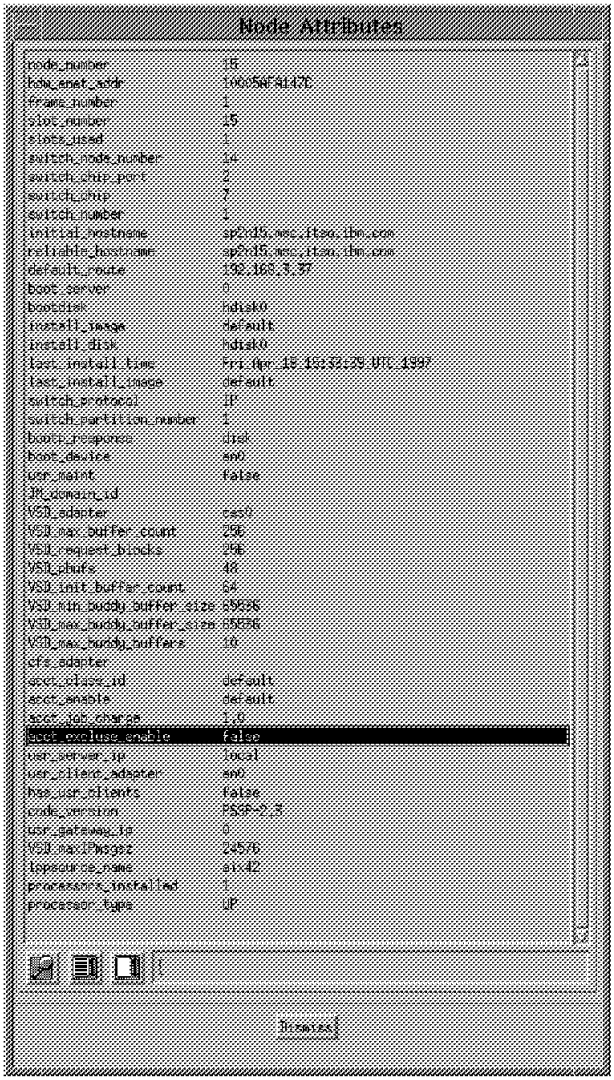


Figure 45. SDR Attributes Displayed for an SP Node. The value field for a selected attribute can be modified.

2. A short path to run commands on the node can be launched directly off the properties dialog when the user clicks on the **Run** button, as shown in Figure 46.

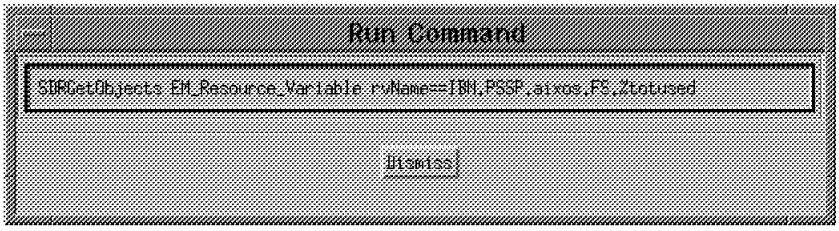


Figure 46. Running a Command in an SP Node

When a managed node has been defined to be a sp_cws, its Properties dialog offers extended capabilities, as shown in Figure 47 on page 111.

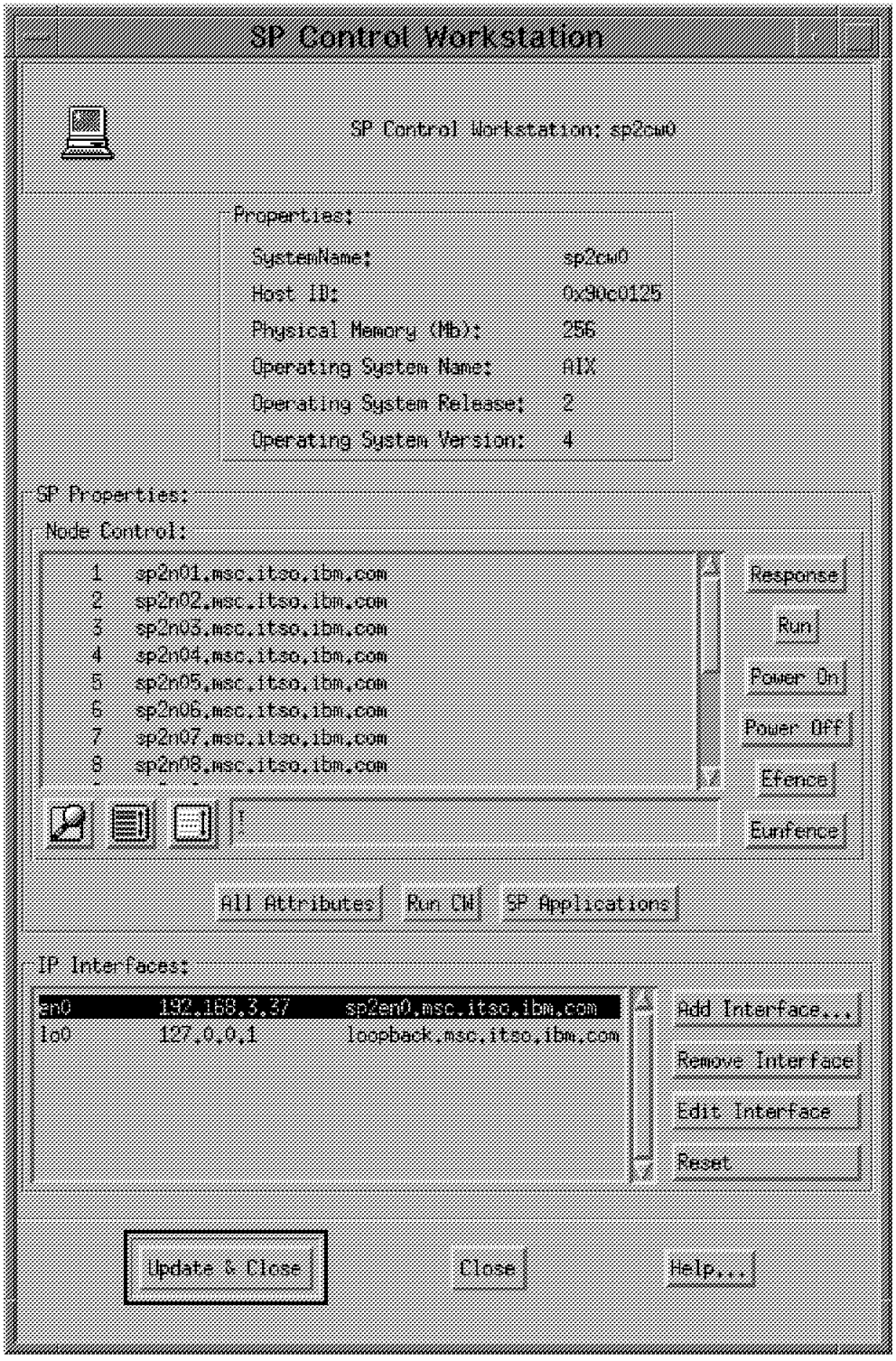


Figure 47. Control Workstation Properties List

The following behavior is available for the control workstation objects:

1. Its System Repository data can be displayed and changed (provided the administrator has the authorization role of super). This is accomplished through the **All Attributes** button. This dialog is similar to the dialog shown in Figure 45 on page 110.

2. A short path to run commands on the control workstation can be launched directly off the Properties dialog (in the **Run CW** button).

3. A node control panel is displayed to perform basic management operations on any of the SP nodes that depend on the control workstation (see Figure 48). Note that all SP nodes are displayed, whether they are also Tivoli managed nodes or not. Once some nodes are selected, the **Response** button provides an extended list that indicates whether each selected node responds and whether they are Tivoli managed nodes.

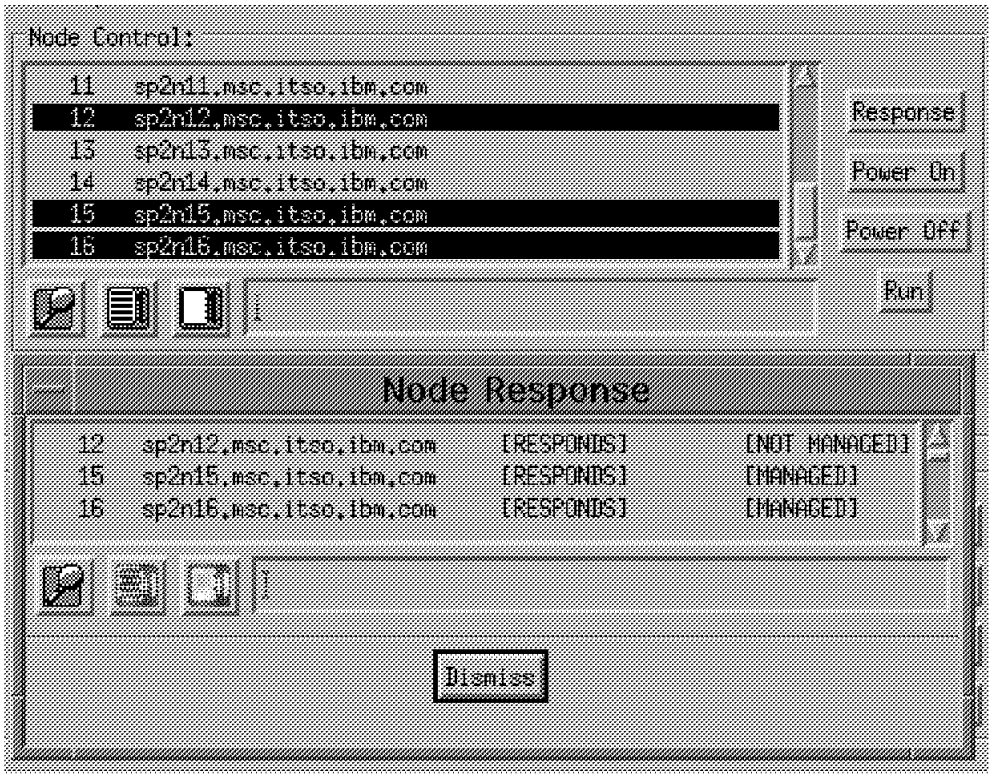


Figure 48. Node Responds Information

4. Any set of nodes from the Node Control list can be selected to perform power on/off operations or to run a command concurrently against such nodes (through the corresponding **power on** and **power off** buttons). Similarly, fence and unfence operations can be performed against the set of selected nodes.

5. A new dialog is provided to serve as a launch pad for RS/6000 SP applications. In the sample customization, the main panels of Perspectives and the System Monitor GUI can be reached directly from Tivoli. This panel, shown in Figure 49 on page 113, appears when the user clicks on the **SP Applications** button in the properties dialog of the SP control workstation. The launch pad can be easily modified by the customer to add or delete new applications.

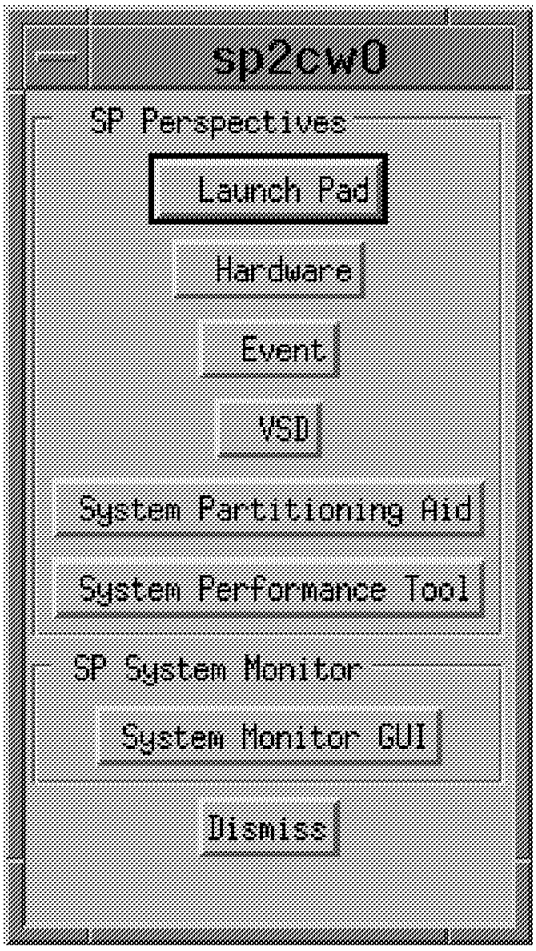


Figure 49. Launch Pad for SP Applications

Unless mentioned otherwise, all the customized SP system management capabilities can be utilized by administrators with authorization roles of super or senior. These permissions can be easily changed by editing the installation scripts that we describe in the following sections.

Our proposed set of customizations are provided with this book. Users that are not interested in providing their own customizations can simply install our customizations, as explained in the next section, and can safely skip the rest of this chapter.

Those interested in understanding how our AEF customizations were made, and wishing to customize their TME 10 Desktop quickly, should read the entire chapter, whose second part provides a basic primer on AEF development.

First, we will go through the installation procedure.

6.2 Installation of the Sample Customizations

Our sample customizations can be installed or uninstalled following the simple procedures that we list in this section. We have provided several Korn shell scripts to perform the basic installation of our customizations, and they can be taken as a model for users who want to install their own.

Before proceeding to the installation, the software has to be downloaded and the appropriate environment variables set. We assume that the user has a Korn shell available to do the following:

1. Login to the Tivoli server machine as root.
2. Create a /.netrc file which allows root access to the Tivoli server machine. It should be owned by root, with file permissions of 600, and contain the following stanza:

   ```
   machine <tivoli server ipaddress> login root password <root's password>
   ```

   so if the server ip address is tivserver.domain.widgetco.com, and root password is mypasswd we would have:

   ```
   machine tivserver.domain.widgetco.com login root password mypasswd
   ```

3. Establish the customization directory. For example:

   ```
   mkdir /tivoli/spcust
   ```

   Note: The following steps assume that the customization directory is /tivoli/spcust.

4. Copy the SPCustomizations.tar file from the diskette distributed with this redbook to the customization directory.
5. Extract the customization environment:

   ```
   cd /tivoli/spcust
   tar -xvf SPCustomizations.tar
   ```

   This creates a directory called Resource in the current working directory (/tivoli/spcust).

6. Customize the Resource/bin/tivoliPaths file (under /tivoli/spcust):
   - SPAEF_TOP should point to the customization directory. For example:

     ```
     export SPAEF_TOP=/tivoli/spcust
     ```

   - SPAEF_SERVER should be the IP address of the server machine. This value must be identical to the value of the Tivoli server ip address previously set in the /.netrc file. For example:

     ```
     SPAEF_SERVER=tivserver.domain.widgetco.com
     ```

   - Alter, delete, or create command line aliases as desired.
7. Execute the script tivoliPaths in the current shell.

   ```
   . Resource/bin/tivoliPaths
   ```

   Now everything is ready for the installation of the customizations. You have to make sure that both the TME server and the managed nodes to be customized are up and running correctly.

8. Edit $SPAEF_TOP/Resource/ManagedNode/LIST and add the list of managed nodes that you want to customize as SP objects. The format of the file is:

   ```
   [#]<sp_cws|sp_node> <ManagedNode Instance Name>
   ```

   The pound sign (#) indicates a comment. That line will be ignored by the installation script.

   Suppose that we want to customize the node sp2cw0.msc.itso.ibm.com as an sp_cws, and nodes sp2n15.msc.itso.ibm.com and sp2n16.msc.itso.ibm.com as sp_nodes. These are the names of the managed nodes as listed in Tivoli.

   The file would look like the following:

```
sp_cws sp2cw0.msc.itso.ibm.com
sp_node sp2n15.msc.itso.ibm.com
sp_node sp2n16.msc.itso.ibm.com
```

9. Execute either spcustomFastInstall or spcustomCustomInstall to install all the customization items. Each script performs essentially the same function. The scripts install all the customization items (callback scripts, dialogs, icons, message catalog) for the managed nodes indicated in the $SPAEF_TOP/Resource/ManagedNode/LIST file. This is accomplished by invoking Tivoli AEF commands described in the following sections.

   The difference between the two scripts is that spcustomFastInstall installs the source files (dialogs, callbacks, and so on) that we have provided. This option is used in those installations running TME 10. However, the spcustomCustomInstall script first extracts the properties_dialog and the icon bitmap and menu for the aix4-r1 managed node from the current Tivoli installation. This script then directs the user to manually insert our customization block into those source files.

   Users with a standard TME 10 installation should use spcustomFastInstall, since it is quick and simple.

   Users who have already customized dialogs for managed nodes should use spcustomCustomInstall. Users with other versions of the Tivoli environment may have a properties_dialog of the managed node does not look like the one in TME 10, and they should also use spcustomCustomInstall. The customization of parent_dialog comes in a block of code that the user can manually insert in their version of parent_dialog following the instructions of the script. All the other dialogs are created for our customization, and can be installed as provided. Once the parent_dialog is modified, the customization process proceeds as in spcustomFastInstall.

   The spcustomCustomInstall script can be used as a model for future customizations in your environment.

   To uninstall the sample AEF/DSL customizations for the SP, simply execute the script spcustomUninstall in the distribution directory.

6.3 What is AEF?

Application Extension Facility (AEF) is a development toolkit provided by Tivoli for the extension of the behavior and data attributes of the TME 10 Desktop objects. These modifications can be taken advantage of from the Tivoli GUI by including references to these methods in the TME 10 Desktop dialogs, which can also be modified by the user.

In order to understand the scope and characteristics of an AEF customization, we need to briefly discuss the basic architecture of an object in the TME 10 framework.

Classes, and instances of those classes (or in Tivoli's terminology, *resource types* and *resource instances*) are represented in a similar way: as objects, that is, entities with data attributes and methods (operations).

Objects representing classes contain references to their class instances. For example, there is an object that represents the managed node resource and provides all services that are common to all managed nodes. This object also

contains references (or pointers) to the managed node instances in the Tivoli environment.

A resource or resource type is a composite of three basic objects: the behavior object (which groups the standard methods of the resource), the presentation object (which contains dialogs and bitmaps associated with the resource), and the extension object (where user customizations are placed).

The first two objects (behavior and presentation) describe the standard behavior of a Tivoli framework object and cannot be modified.

The extension object, however, is a placeholder for new methods, attributes, or dialogs that users might want to add to an already defined resource or resource type. You can think of the extension object as a built-in framework mechanism that allows the quick extension of object behavior without reverting to standard object-oriented methods like inheritance, which can be costly and complex in a CORBA environment like Tivoli.

When a service (a method, an attribute, a dialog, a bitmap, and so on) is requested for an object, the Tivoli run time system looks in the extension object to see if that service is defined there. Only if it is not, the other two objects are searched. This way, the original behavior of an object can be extended without altering its original contents. In Figure 50, the structure of a framework object is shown. Again, this framework object can represent a resource type (a class) or a resource instance (an instance). The extension object is initially empty.

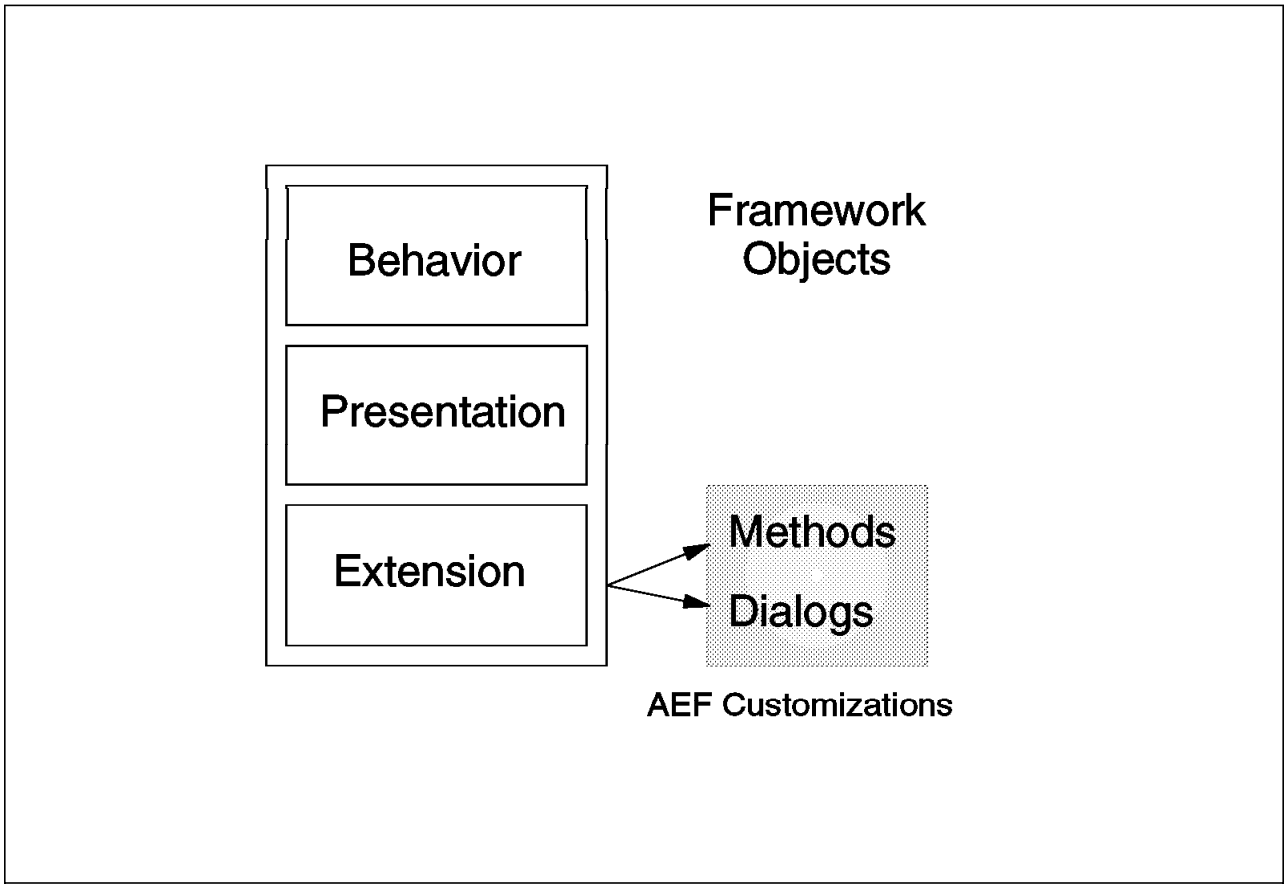


Figure 50. The Framework Object. AEF modifies the extension object.

Methods can be added and deleted from the extension object freely, without affecting the original behavior of the resource type or resource instance. That is, AEF customizations can be made both at the class level or at the instance level and do not erase the original contents of an object.

Once objects have been extended, the TME 10 Desktop dialogs can be modified to take advantage of the new methods and attributes.

Tivoli dialogs are written in the Dialog Specification Language (DSL) and they are installed in the objects as if they were attributes.

- Dialogs installed at the class level can be used by any instance of the class.
- Dialogs installed at the instance level, override the global dialogs.

For example, in our sample customizations, we have customized the Properties dialog (in Figure 44 on page 109 and Figure 47 on page 111) just for certain instances of the managed nodes. Hence, these dialogs are installed at the instance level.

It is worth noting that DSL development (which only affects the TME 10 Desktop dialogs) is not, strictly speaking, part of AEF development. However, the customization of dialogs is an integral part of AEF development. Hence, they are always discussed together.

AEF/DSL development can be used in general for many purposes:

- Addition of methods (callbacks in AEF terminology) to both resource types and resource instances.
- Addition of attributes to resource instances.
- Modification of the existing dialogs in the TME 10 Desktop.
- Addition of new dialogs to the TME 10 Desktop.
- Customization of bitmaps and icons, which are the representation of resources in the TME 10 Desktop (an icon is defined as a pair consisting of an icon and its associated pop-up menu). See Figure 51 on page 118.

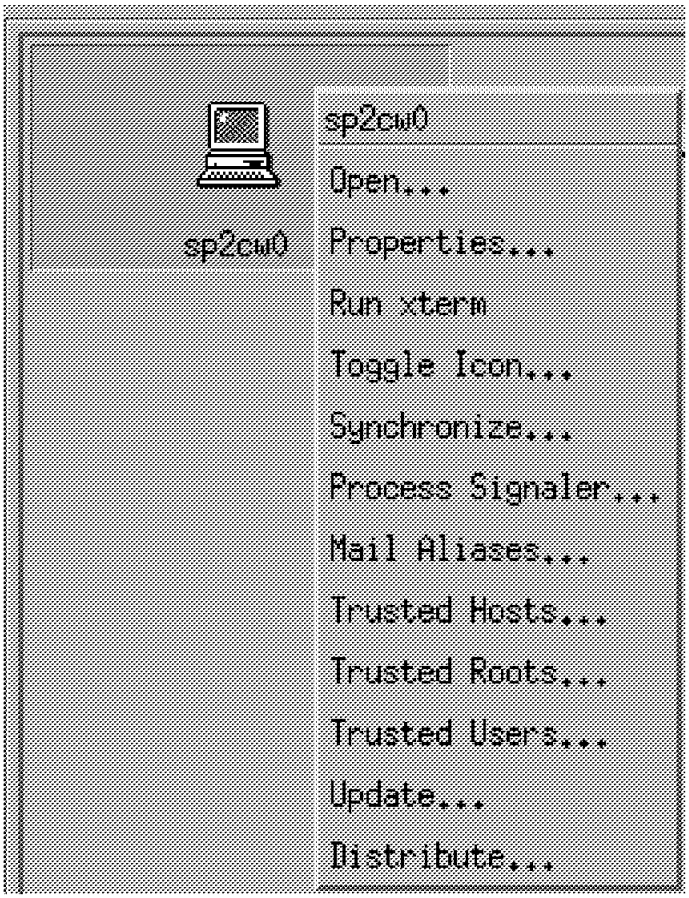


Figure 51. An Icon with its Associated Bitmap

- Definition and installation of message catalogs associated to a set of dialogs.

All these customizations are done at run time, through the command line interface, without the need to reload the database or restart the TME server.

6.3.1 Strengths and Weaknesses of AEF/DSL

It is worth noting that modifying the behavior of the TME 10 Desktop through AEF/DSL is considerably simpler than using the Advanced Development Environment provided by Tivoli (ADE/Tivoli).

ADE/Tivoli requires proficiency in Interface Definition Language (IDL), which is the language used to define objects in a standard CORBA system, and TEIDL (Tivoli Extension IDL).

TEIDL is an extension of IDL including, among other features, exceptions (similar to those of C++) and object-naming facilities. All Tivoli resources are defined in TEIDL, and ADE development extends the original framework seamlessly. However, ADE development is complex and requires a substantial investment in development resources.

Since AEF/DSL provides a simple, yet powerful, mechanism to extend the behavior of objects in the TME 10, we believe that most users can benefit from this type of development in order to obtain an integrated management environment for the SP. As we saw in the previous section, the behavior of

managed nodes can be easily extended to provide virtually any functionality desired by the user.

However, it is important to remember that even though AEF/DSL facilitates the customization of the TME 10 Desktop, it is not intended to be a complete development facility for the TME 10. Only the ADE fulfills this role. Thus, the reader should also be aware of the inherent limitations of AEF/DSL customizations:

- Preexisting object methods and attributes cannot be edited or modified.
- Objects and dialogs are updated through commands issued at run time. These changes are permanently registered in the TME database, and only take effect when the TME 10 Desktop is restarted.
- In the TME 10 Desktop, object methods are invoked as callbacks associated with gadgets in the context of the object who owns the dialog. Other objects′ methods cannot be directly called (this type of call can only be issued through ADE).
- Drag-and-drop operations, as defined in the TME 10 Desktop, cannot be modified.
- Strictly speaking, not all dialogs can be effectively modified. For example, the create dialogs (dialogs used to create new instances of a resource from the TME 10 Desktop) cannot be modified in the case of a managed node. The reason for this is that when a managed node is created, its client software has to be installed. This is a complex process and it should not be tampered with.

For a complete list of the capabilities of AEF, refer to the *TME 10 AEF User′s Guide*.

6.4 Anatomy of an AEF/DSL Customization

An AEF/DSL customization involves the addition of methods or attributes to an object and the modification or creation of its dialogs in order to take advantage of the new behavior. This section discusses the basic mechanism of the customizations. The reader not interested in AEF development can safely skip the rest of this chapter.

Even though the customization process is not complex, it does involve several steps, which can be better illustrated if we go through an example. Throughout the rest of the chapter we will refer to the customization of a managed node as an SP control workstation (the customization of SP nodes is even simpler). In our environment, the control workstation′s name is sp2cw0.

As we saw in Figure 47 on page 111, the Properties dialog (named properties_dialog) of the managed node has to be modified to produce the extended dialog shown there. This dialog contains buttons (**Run CW**, **SP Applications**) that open new dialogs, specific to our sample customizations. These dialogs were newly created and installed in the managed node resource.

Usually, the first step is to look at the dialogs that are currently installed in a resource by issuing the `wlsdialog` command. An example of the results of this command for a managed node is shown in the following screen:

```
wlsdialog -r ManagedNode

dialog name (customization status)

view_dialog (resource-wide customization)
view_dialog (resource-wide customization)
view_dialog
parent_dialog
add_dialog
...
```

The output of wlsdialog lists the dialog names and whether they are the product of customizations or not. In the preceding example, add_dialog and parent_dialog are standard dialogs, as well as the second occurrence of view_dialog. The first occurrence of view_dialog is a resource-wide customization.

As we mentioned before, AEF/DSL customizations add items (dialogs, methods, icons, and so on) to Tivoli objects, never affecting the original contents of the object (in this case, the managed node resource type). This is why customized dialogs appear twice in the listing.

Once we have identified the dialog we want to modify (in this case, parent_dialog), we can extract it as follows:

```
wgetdialog -r ManagedNode parent_dialog >parent_dialog.d
```

Dialogs are stored in binary. The output file can be given any name, but it should have a suffix of .d. The contents of parent_dialog.d must be reverse-compiled in order to obtain an editable file.

```
rdsl parent_dialog.d >parent_dialog.dsl
```

The file parent_dialog.dsl now contains the source code for the standard parent_dialog of the managed node resource. This file can be edited, modified, or augmented with new gadgets that invoke the methods of managed nodes (we provide more details on DSL dialogs in 6.5, “TME 10 Desktop Dialogs” on page 123). For the time being, suppose that parent_dialog.dsl contains all the customizations that we have developed.

In order to install the dialog in the extension object of the managed node, enter the commands as shown in the following example:

```
#COMPILE THE SOURCE DIALOG
dsl parent_dialog.dsl >parent_dialog.d
#INSTALL IT AT THE RESOURCE LEVEL
wputdialog -r ManagedNode parent_dialog <parent_dialog.d
```

Now the dialog will appear whenever the managed node parent_dialog is invoked. However, this is not exactly what we want, for not all nodes in our installation will be SP control workstations.

We want this new parent_dialog to appear only in a specific node in our installation, say sp2cw0. (We have shown the preceding example just as an illustration of a resource-wide customization.) The real invocation of the wputdialog command should be as follows:

```
wputdialog -l @ManagedNode:sp2cw0 parent_dialog <
parent_dialog.dsl
```

This is how instances are referenced within AEF commands, with syntax @ResourceType:resourceInstance. Now the customized parent_dialog will appear *only* when the parent_dialog of sp2cw0 is invoked.

Now, the dialog is installed and ready to use. Use the `wlsdialog` command to verify that it is correct, as shown in the following example:

```
wlsdialog -r ManagedNode

        dialog name (customization status)

        view_dialog (resource-wide customization)
        view_dialog
        parent_dialog (customized for sp2cw0)
        parent_dialog
        add_dialog
        ...
```

Usually, your new dialogs also reference new *methods* of the resource. For example, for our customizations of the parent dialog, we needed a list to display the node numbers of all the nodes controlled by the control workstation (the list that appears under the Node Control section in Figure 47 on page 111). This method is invoked in the parent_dialog (the details of the method invocation and implementation will be discussed in a later section), and it has been implemented in the script sp_cws.GetNodeNumbers.ksh (written in Korn shell).

In this case, it makes sense to install the method as a resource-wide customization, for it is only used from the customized parent_dialog. This way we avoid having to install it remotely in the nodes customized as control workstations (which is what would happen if we installed the method on a per instance basis). Only one copy of the method in the TME server is necessary in resource-wide customizations.

Let us walk through the case of installing the method implemented by the script sp_cws.GetNodeNumbers.ksh.

If this is the first new callback that is installed in our environment, we must create a new directory, called /$INSTALL_DIR/bin/$interp/CUSTOM, where INSTALL_DIR is the path of the TME installation directory and $interp is the platform for which the script was written (in our case, the value of $interp is aix4-r1).

If the callback is intended for a resource-wide customization, this directory is in the server file system. If the callback is intended for just an instance of a managed node, then this directory should be created where the Tivoli local database has been installed for the managed node.

The callback implementation file is placed in this directory with permissions 755.

The installation of a new callback is done through the command line, using the `wputmeth` command, as shown in the following example:

```
wputmeth -i aix4-r1 -a user -a super -a senior -a admin -r ManagedNode
sp_cws.GetNodeNumbers /aix4-r1/CUSTOM/sp_cws.GetNodeNumbers.ksh
```

Note that the name of the method and the script that implements it do not need to match, though we have used the name of the callback with a suffix of .ksh for our scripts, for the sake of simplicity.

We need to say a few words about this `wputmeth` call. The method has been installed for the aix4-r1 platform (Tivoli does not yet support AIX 4.2) as a resource-wide customization of the managed node. The -a flag specifies authorization roles for the method.

In the preceding example, all user roles are granted privileges to run this method, since any user that can access this dialog should be able to display it (otherwise, the parent_dialog would give an error indicating the method was not found). In this case, the access control information is not necessary, for if no access privileges are specified, any Tivoli user with access to the resource can execute the method. If you wish to change the access privileges to a custom method, you must uninstall it first (using `wrmmeth`) and reinstall it with the new privileges.

Note finally that the method path (/aix4-r1/CUSTOM/sp_cws.spmon.ksh) is a relative path with respect to $INSTALL_DIR/BIN.

Furthermore, we could have installed this method as part of an instance, say sp2cw0, by using `-l @ManagedNode:sp2cw0` instead of `-r ManagedNode`.

Once `wputmeth` has completed successfully, the sp_cws.GetNodeNumbers method becomes part of the behavior of managed node resources. We need to repeat the process for each new callback that our customized parent_dialog uses.

Once the new version of parent_dialog and its associated callbacks have been installed, the customization can be tested. After restarting the TME 10 Desktop, whenever the parent dialog of sp2cw0 is invoked, we should see the new customized dialog, seamlessly integrated with the rest of the Tivoli environment.

This is how a basic AEF/DSL customization takes place. Of course, we have left out many important details. More information about the commands used here is available in the *TME 10 AEF User's Guide* or in the Tivoli man pages. Also, we need to know how to modify dialogs and how to construct the callbacks. This will be covered in the next sections.

The reader has probably noticed from the figures in 6.1, “High Level Overview of a Sample Set of Customizations” on page 107 that, in addition to the new dialogs, we have also introduced new bitmaps for our customized nodes. Tivoli icons can be modified so that managed nodes (or any other resources, like policy regions or profile managers) can be visually distinguished in the TME 10 Desktop.

Before going into more details on AEF/DSL, we need to know how to remove an unwanted customization.

6.4.1 Removing AEF/DSL Customizations

AEF/DSL customizations can be easily removed from your environment without affecting the standard behavior of the Tivoli resources. Suppose we were not satisfied with the customization of the parent_dialog and its associated method sp_cws.GetNodeNumbers in the previous section. We can remove the dialog by issuing:

```
wrmdialog  -l @ManagedNode:sp2cw0 parent_dialog
```

If the dialog was installed as a resource-wide customization, then we would have used, instead of the instance name, `-r ManagedNode`.

It is equally easy to remove a callback. Again, we use the -l flag to remove instance customizations and -r to remove resource-wide customizations. In the case of sp_cws.sp_applications, we issue:

```
wrmmeth -r ManagedNode sp_cws.sp_applications.
```

The customizations no longer exists and sp2cw0 should now behave like a standard AIX managed node.

6.5 TME 10 Desktop Dialogs

Since AEF customizations are tailored to alter the behavior of the TME 10 Desktop, it is necessary to get aquainted with the Dialog Specification Language (DSL), as an integral part of the customization process.

A full discussion of DSL is given in *TME 10 AEF User's Guide*. We aim, in this section, to provide enough background to carry out simple, yet meaningful, customizations to the existing TME 10 Desktop dialogs.

In order to illustrate the process, we will work with the simplest of our dialogs. This dialog offers a set of buttons to launch several SP applications, like SP Perspectives and the SP System Monitor GUI, from an SP control workstation object. This is illustrated in Figure 52 on page 124.

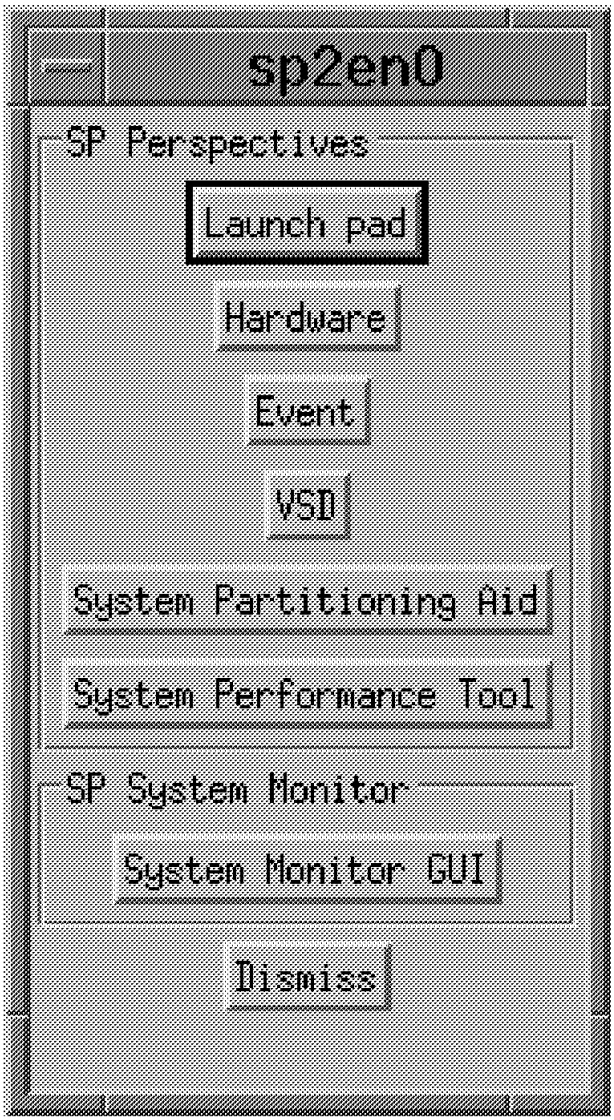


Figure 52. SP Applications for the Control Workstation

This dialog has been created for our sample customizations and it is installed following the procedure discussed in the last section. It appears when the user clicks on **SP Applications** on the Properties dialog. We will not concern ourselves with how the launch pad can be invoked from parent_dialog for now. Let us concentrate only on its dialog definition.

The source code of this dialog starts as follows:

```
Command Dialog
{
   Variables
   {
      CString man_node_name;
   }

   Attributes
   {
      Name = sp_launch_applications;
      Title = Msg(SpManagedNode, $man_node_name, 1, $man_node_name);
   }

   Group
   {
      Attributes
      {
         Name = g1;
         Title = Msg(SpManagedNode, "SP Perspectives", 2);
         TitlePos = TOP;
         Border = YES;
         Layout = VERTICAL;
         ChildColumnAlignment = STRETCH;
      }
      Gadgets
      {
         Button
         {
            Name = launch_pad;
            Title = Msg(SpManagedNode, "Launch Pad", 3);
            Commands = sp_cws.LaunchPerspectives@("perspectives");
         }

...
      }
...
   }
...
}
```

This definition covers up to the first button (**Launch Pad**). The rest of the dialog is a set of similar button definitions, and is provided in Appendix H, "Contents of AEF Customization Scripts" on page 201. Here we will use these lines to analyze the structure of a dialog.

A DSL specification consists essentially of a list of the gadgets (buttons, in the case of the sp_cws.sp_applications dialog) of a dialog. Furthermore, each gadget is associated to one or more actions that allow the administrator to send messages from the dialog to its corresponding resource. In our example, the command sp_cws.LaunchPerspectives (the last code line in the example) is a method (a callback) of the control workstation object that launches the SP Perspectives GUI. For the installation process of a callback, see 6.4.1, "Removing AEF/DSL Customizations" on page 123.

6.5.1 Basic Structure of a DSL File

A DSL dialog has a very simple structure. It is made up of *blocks*, which are sets of statements enclosed within curly braces enclosed in turn in a main dialog block (starting with the keywords Command Dialog). Every statement in a block is terminated by a semicolon. Blocks can appear in any order in the file, and can occur any number of times within the same file.

There are three types of blocks in DSL, each with a defining keyword: Attributes, Variables, and Gadgets.

There are also *group blocks*. These group blocks can contain attribute, variable, or gadget blocks, as in the case of the sp_cws.sp_applications dialog previously listed.

We will take a brief look at these three types of blocks in the next sections.

6.5.2 Variables and Variable Blocks

Variables and variable blocks contain a set of variable declarations. DSL supports three data types:

String Similar to a char* in C.

CString A data type encoded within a string.

CMsg A data type encoded as a message.

A variable is declared with the following syntax:

```
type variable_name =  variable_value;
```

The variable value could be either a constant or a method invocation (in other words, a callback).

When a variable is not assigned any value from the dialog itself, its value is assumed to proceed from the calling dialog, as a parameter of the dialog invocation. We will call these variables *dialog parameters*. We cover the details of dialog invocation in the section 6.6, “Methods for Customized Objects” on page 129.

In our preceding example, we have defined the variable man_node_name, which is a CString. That is, when the dialog is invoked from a callback, the value for this variable is provided (more details on this later). Note that when we reference this variable, we prefix it with the dollar sign ($), as in the following function call:

```
Title = Msg(SpManagedNode, $man_node_name, 1, $man_node_name);
```

There are also some predefined variables, called *desktop variables*, which are available in any dialog. The most important desktop variables are:

$owner The object reference of the dialog owner; that is, the object that the dialog belongs to (used as a callback argument).

$selects The object references of the objects that have been selected in a collection of icons in the dialog.

$icon The object reference of the resource represented by an icon.

$self The identifier of the dialog (each dialog instance has associated with it an instance ID, which is stored in this variable).

As an example of the use of one of these desktop variables, let us list the definition of the **Dismiss** button in our sample dialog. This button is associated to a library call, dtc_dismiss, that requires as a parameter the identifier of the dialog to be dismissed (that is, the sp_cws.sp_applications itself), as shown in the following example:

```
Button
{
   Name = dismiss;
   Title = Msg(GenericCollectCat,Dismiss,40);
   Commands = dtc_dismiss($owner,sp_launch_applications, $self);
}
```

Constant strings are enclosed in double quotes (″). Comments are allowed and they are enclosed in the sequences ′/*′ and ′*/′. They can expand any number of lines, but they cannot be nested.

6.5.3 Attribute Blocks

An attribute block is a list of attribute names and values. Each pair represents an attribute of the dialog or a gadget. In our example, the following block defines the name of the dialog (sp_launch_applications) and its title, which is the name of the managed node (this is the title of the window the user sees):

```
Attributes
{
        Name = sp_launch_applications;
        Title = $man_node_name
}
```

If the control workstation is spcw0, we could have hardcoded this name in the dialog:

```
Title=″spcw0″;
```

However, a more elegant approach is to use the value of $man_node_name, which contains the name of the current managed node.

Attributes can appear in any order in DSL and all are optional, with the exception of the Name attribute.

The attribute block of the example corresponds to the complete dialog, but there can be attribute blocks specific to gadgets.

For example, the list gadget can have attributes like Visible (to indicate whether the list is visible in the dialog), Sensitive (to indicate whether the list is sensitive to user input) or Rows (the number of text rows to display).

As you would expect, some attributes are specific to a gadget. For a complete list of the gadgets that can be defined in DSL, together with their specific attributes, see the *TME 10 AEF User′s Guide*. Information on gadgets is also available through the standard UNIX manual pages facility when AEF has been installed in your system.

6.5.4 Gadget Blocks

A gadget block contains a list of gadgets, such as Table, List, Button, CommandButton, and so on. The definition of the gadget itself is similar to the block definition. Again, we follow the example of the sp_cws.sp_applications dialog, as shown in the following example:

```
Gadgets
{
   Button
   {
      Name = launch_pad;
      Title = Msg(SpManagedNode, "Launch Pad", 3);
      Commands = sp_cws.LaunchPerspectives@("perspectives");
   }

   Button
   {
      Name = hardware;
      Title = Msg(SpManagedNode, "Hardware", 5);
      Commands = sp_cws.LaunchPerspectives@("sphardware");
   }
...
}
```

In this gadget block we have defined two buttons; the first one invokes the SP Perspectives launch pad, and the second one directly invokes the SP Perspectives hardware view.

Each gadget must have a name attribute that identifies it uniquely within the dialog. Also, gadgets like button have a title attribute, which is the name that appears on the button in the TME 10 Desktop. In this case, we have used the message catalogs facility in order to provide some internationalization support in our messages. For now, the reader can think of the value of the title attribute as a string, as if we had typed

```
Title = "Launch Pad";
```

or

```
Title = "Hardware";
```

The commands attribute references the callback associated with the gadget. In this case, both buttons reference the same callback, sp_cws.LaunchPerspectives. We do not need to concern ourselves with the details of the method call itself in this section.

Note also that we do not need to specify the physical position of a gadget in a dialog display. DSL determines the physical layout of the display, arranging gadgets according to their order in the file (from left to right, from top to bottom) unless otherwise specified by the developer.

Gadgets can be grouped in a Group{} block. Every group of gadgets has its own attributes block, and it can serve to identify visually a group of gadgets. We have used this facility in our sample dialog to separate the buttons that launch SP Perspectives views from the button to invoke the SP System Monitor GUI, as shown in the following screen:

```
Group
{
   Attributes
   {
      Name = g1;
      Title = Msg(SpManagedNode, "SP Perspectives", 2);
      TitlePos = TOP;
      Border = YES;
      Layout = VERTICAL;
      ChildColumnAlignment = STRETCH;
   }
   Gadgets
   {
      Button
      {
         ...
      }
      Button
      {
         ...
      }
      ...
   }
}
```

In the preceding example, the button definitions are displayed within a frame. The frame is determined visually by a border (indicated by the attribute Border = YES). The buttons are placed vertically (Layout = VERTICAL) within the frame, filling the space allocated for the group (ChildColumnAlignment = STRETCH).

6.6 Methods for Customized Objects

Callbacks are method invocations with two purposes in a DSL dialog: to assign a value to a variable, or to be invoked when a gadget is selected. (They appear as the value of the command attribute on a gadget, as we saw in previous examples). Callbacks used in AEF/DSL are either provided by the developer (custom callbacks) or by the Tivoli libraries.

The callbacks used in DSL dialogs correspond to several types:

- Desktop callbacks, which are library callbacks and handle various aspects of the dialog presentation and gadget manipulation (like dismissing or refreshing a dialog). These callbacks are passed from the dialog to the TME 10 Desktop, as opposed to the callbacks mentioned in the following section, which are external programs to the TME 10 Desktop.
- Legacy callbacks, which are shell scripts (preferably written in Perl, Bourne, or Korn shell) that take arguments on the command line, read from standard input, and write to standard output.
- ADE callbacks, which are identified by a percent sign (%) suffix. These callbacks belong to ADE development and will not be discussed here.

AEF development uses desktop callbacks and legacy callbacks. We cover them in the next section, starting with the desktop callbacks.

6.6.1 Desktop Callbacks

As previously mentioned, desktop callbacks are in libraries provided by Tivoli. AEF makes them available to a dialog automatically. Two libraries are currently provided: desktop command callbacks and gadget library callbacks.

Even though their functionality is similar, gadget library callbacks are more comprehensive, and Tivoli recommends using them instead of desktop callbacks. However, we have used desktop callbacks in our sample customizations, because we have had problems with gadget library callbacks, especially when used in managed node dialogs. Therefore, we have limited our use of desktop callbacks to only dismiss a dialog.

Desktop command callbacks start with the prefix 'dtc'. For example, dtc_refresh instructs the TME 10 Desktop to execute the refresh callbacks for the dialog, and dtc_dismiss dismisses the current dialog.

All callbacks are used similarly. We have already seen an example of a desktop callback in the button that dismisses the launch dialog, as shown in the following screen:

```
Button
{
   Name = dismiss;
   Title = Msg(GenericCollectCat,Dismiss,40);
   Commands = dtc_dismiss($owner,sp_launch_applications, $self);
}
```

This code defines a button that dismisses the window. The desktop command callback is defined as the value of the commands attribute. Its parameters are the predefined variables $owner, $self (defined in 6.5.2, "Variables and Variable Blocks" on page 126) and the name of the current dialog, sp_launch_applications. This is the only library callback that we have used in our sample customizations.

Gadget library calls are identified with an ampersand sign (&). For example, the following call loads the dialog given by descriptor, assigns it a name and loads its variables given in the parameters list:

```
create&(name, descriptor, parameters)
```

Every gadget has also a set of specific gadget library callbacks (for example, set_left_column, set_row_select or set_rows are library calls for the table gadget). As we mentioned in the beginning of this section, we have not used gadget library callbacks in our customizations.

6.6.2 Legacy Callbacks

All other callbacks implemented in our customizations are legacy callbacks. They are used to gather data from the SP system or invoke SP applications. You should think of legacy callbacks as scripts, written in Perl, Bourne, or Korn shell languages, which execute RS/6000 SP commands and relay their output to the appropriate gadget in a dialog.

Our sample customizations are written in Korn shell language. They are invoked from a dialog, just as desktop callbacks are. Again, we follow the example of the sp_cws.sp_applications dialog, as shown in the following screen:

```
        Button {
        Name = system_monitor;
        Title = Msg(SpManagedNode, "System Monitor GUI", 16);
        Commands = sp_cws.spmon@();
        }
```

Legacy callbacks calls are characterized by suffixing the symbol '@' after the callback call. The method sp_cws.spmon is a very simple Korn shell script (sp_cws.spmon.ksh) that invokes the SP System Monitor GUI to be displayed in the administrator's TME 10 Desktop. We list it in the following example:

```
#!/bin/ksh

export DISPLAY=$WD_DISPLAY

# We close stdin, stdout and stderr
exec 0<&-
exec 1<&-
exec 2<&-

/usr/lpp/ssp/bin/spmon -g

exit 0
```

This script raises several important issues about legacy callbacks, and it is worth examining these issues in detail.

First, since we want to be able to launch the SP Hardware Monitor GUI in the administrator's display, we set the DISPLAY variable of the shell executing the script to $WD_DISPLAY. Callback methods receive a set of predefined environment variables from Tivoli. Among them is WD_DISPLAY, which identifies the display of the calling dialog (in this case, sp_launch_applications).

Other important environment variables passed to the shell executing the script are PATH, LD_LIBRARY_PATH, and OWNER.

In the next three executable lines, we close the I/O streams associated with the script. When the TME invokes the callback, it keeps control over the shell I/O streams and all other file descriptors that are opened in the callback. Since we want to be able to launch one application (in this case, the SPMon GUI) and return control to the TME 10 Desktop immediately afterward, we must close all file descriptors associated with the callback.

The last line of the script returns an exit code to the TME 10 Desktop. Callbacks in Tivoli use the traditional error code semantics in UNIX. In general there are three predefined codes that the TME 10 Desktop will understand:

0 Success

1 Error

2 Usage error

As mentioned before, legacy callbacks can be used to perform multiple functions. In our prespecified customizations, we can access (read and/or write) the SP System Data Repository, execute commands against a set of SP nodes or control workstations, or launch specific SP applications.

We also use callbacks to manipulate the TME 10 Desktop itself: posting new dialogs, posting messages, or updating information on a visible dialog. All these TME 10 Desktop management functions are performed through special callback utilities provided by Tivoli which are described in 6.6.3, “Callback Method Utilities.”

6.6.3 Callback Method Utilities

Before returning control to the TME, a legacy callback might need to manipulate the TME 10 Desktop by posting a new dialog or a message. In order to manage desktop dialogs and messages, several CLI commands are provided in AEF. The following list represents those commands that we have found helpful in our development.

- `wpostdialog`

  This command posts a new dialog from a callback and is part of the AEF Command Line Interface. Its syntax is very simple:

  ```
  wpostdialog dialog_name name1 ... nameN param1 ... paramN
  ```

  where name1...nameN are the names of the dialog parameters (without the preceding dollar sign) and param1...paramN are their corresponding values. As a more concrete example, consider the call:

  ```
  wpostdialog sp_cws.sp_applications man_node_name "$1"
  ```

  This call appears in the method sp_cws.LaunchApplicationsDriver which is associated with the **SP Applications** button in the parent_dialog of the control workstation. When this button is selected, the method posts the dialog sp_cws.sp_applications, giving as the value of the variable man_node_name, the first argument received by the script. That is, in the parent_dialog, we have the following:

  ```
  Button
       {
       Name = launch_applications_driver;
       Title =  Msg(SpManagedNode, "SP applications", 23);
       Commands = sp_cws.LaunchApplicationsDriver@($man_node_name);
                                  }
  ```

  This is how we can invoke one new dialog (sp_cws.sp_applications) from a modified one (parent_dialog) and pass variables from one to the other (see the previous examples to find out how the variable man_node_name is declared within the sp_cws.sp_applications dialog). Note that the variable man_node_name is passed *in quotes* to the new dialog.

  The new dialog is made a child of the dialog that invoked the callback, and therefore it becomes a dialog that belongs to the same object (indicated by $owner). The option -t could be used in the `wpostdialog` call to make the new dialog independent of the current dialog.

- `wdispmsg`

  This command displays a new window with a message in the TME 10 Desktop. For example:

```
wdispmsg "$result"
```

We have used it when running a command against a node or the control workstation on the SP in order to display the result of the computation ($result is a variable local to the script).

- wdisperr

  This command is very similar to wdipsmsg, but it presents the window as an error. Consider the example:

  ```
  wdisperr "Operation could not be completed"
  ```

- wdispconf

  This command displays a confirmation window, asking the user to make sure that critical operations should indeed be executed. We have used this command to display a confirmation message after the user has issued a power off request against a set of nodes:

  ```
  ANSWER=wdispconf "Are you sure you want to power off nodes $* ?"
  ```

  The user will be prompted to enter YES or NO at the new window where the message is displayed. The return value of wdispconf is this answer. There is also a more general form of the wdispconf command where the user can indicate the labels of the YES or NO buttons:

  ```
  ANSWER=wdispconf -y "Cancel" -n "Continue"
  "Do you want to continue the operation?"
  ```

- wstatusline

  This command displays a message on the status area of the main window in the administrator's TME 10 Desktop. For example:

  ```
  wstatusline "nodes $* have been powered off"
  ```

- wgetvalue and wsetvalue

  These commands get and set the values of a dialog gadget. The following example is a simplification of the actual use of wgetvalue in our customizations:

  ```
  wgetvalue -o $OWNER -d $DIALOG -i $INSTANCE
  $GADGET "new_value"
  ```

6.7 Bitmaps

Bitmaps are a very effective way to present information in a GUI. Tivoli associates bitmaps with all its managed resources. These bitmaps appear often in the dialogs belonging to the resource. When a bitmap represents a resource instance in the TME 10 Desktop (that is, it appears in the TME 10 Desktop and its pop-up menu is available), then it is called an icon. We will cover icons in the next section.

Bitmaps are stored in the X pixmap (XPM) format. There is a public domain XPM editor included with the Tivoli Productivity package.

To list the bitmaps currently associated with a resource, issue:

```
wlsbitmap -r ResourceType
```

Suppose that we have already created our own bitmap, or modified any of the bitmaps already available in the resource. The bitmap can now be installed at the instance level, or as a resource-wide bitmap.

In our sample customizations, we have created a new icon for managed nodes to be represented as SP nodes. The bitmap name is sp_node.xpm, and we have installed it using:

```
wputbitmap -r ManagedNode sp_node <sp_node.xpm
```

That is, the name of the bitmap is sp_node. This bitmap can be now used in managed node dialogs. Also, a bitmap can be installed for a specific instance of a managed node using the -l flag, as in the commands `wputdialog` and `wputmeth`, explained in 6.4, “Anatomy of an AEF/DSL Customization” on page 119.

A bitmap can be removed using `wrmbitmap`. If we want to delete our previous customization, we would issue:

```
wrmbitmap -r ManagedNode sp_node
```

6.7.1 Icons

You can think of an icon as a pair made up of a bitmap and a pop-up menu, which is defined as a DSL dialog. Icons are treated as special entities that identify a resource. In this sense, the icon can be viewed as a *state* for a resource. Icons are installed at the resource level, and each instance is in a given state when it appears in the TME 10 Desktop.

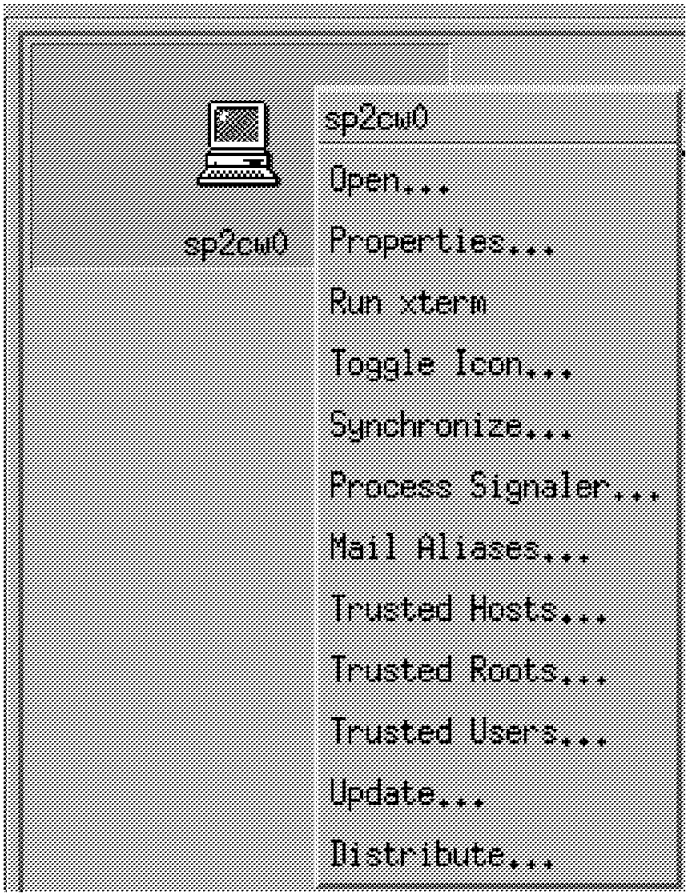
The idea of the state of a resource makes sense when thinking about managed nodes and their different platforms. For example, in the standard Tivoli installation, these are some of the states for a managed node resource:

sunos4-client
sunos4-server
solaris2-client
solaris2-server
aix4-r1-client
aix4-r1-server

Most other resources (policy region, profile manager, and so on) have only one state: normal.

In order to list all the states of a resource, say, a managed node, we use the command `wlsicon`, as follows:

```
wlsicon -r ManagedNode
```



and its associated pop-up menu

Figure 53. The Control Workstation Icon, which is a bitmap

New states can be added to a resource through the `wputicon` command, which associates a bitmap with a pop-up dialog. In the case of our sample customizations, we have used the same bitmap that was installed in the previous section. However, if the user wants to retrieve the existing bitmap for an icon, `wgetbitmap` with an option indicating the state of the resource must be used. In the case of the managed node, issue the following:

```
wgetbitmap -I aix4-r1 -r ManagedNode >cust.xpm
```

Similarly, the pop-up menus must be retrieved using the state name:

```
wgetdialog -I aix4-r1 -r ManagedNode >cust.d
```

The pop-up menus are DSL-defined dialogs, like all other desktop dialogs. To install the icon, issue the following:

```
wputicon -r ManagedNode sp_node  sp_node.d sp_node.xpm
```

In order to remove the sp_node icon, we issue:

```
wrmicon -r ManagedNode sp_node
```

A resource instance has a state. This state can be changed through the command `wputstate`. If we had a managed node (sp2n1.msc.itso.ibm.com) that we want to customize as an sp_node, we would run:

```
wputstate -l @ManagedNode:ps2n1.msc.itso.ibm.com sp_node
```

As expected, `wputstate` can only be run as an instance.

Note: States are only used when referring to icons. They cannot be used when installing other dialogs.

The reader has probably noticed that customizations of the parent_dialog for the control workstation (see 6.4, “Anatomy of an AEF/DSL Customization” on page 119) were made at the instance level or at the resource type level, not at the state level.

6.8 Messages and Message Catalogs

Tivoli offers a message catalog facility, where the text appearing in the dialog can be separated from the dialog definition itself and stored in a message file. This provides basic internationalization support.

In our experience, message catalogs are cumbersome to use. They should only be taken advantage of in extensive customizations, or when internationalization support becomes critical. Readers who are not interested in providing internationalization support can safely skip this section.

Continuing the approach used in this chapter, we discuss message catalogs through an example taken from our customizations. We have included the text of our customized dialogs in a new message catalog called SpManagedNode. To build the message catalog, we create the file SpManagedNode.msg that contains all the text messages that users see in dialogs, using the following format:

```
$set 2
$ g1_Title
1   SP Perspectives
$ launch_pad_Title
2   Launch Pad
...
```

This catalog is compiled using the gencat facility provided with the AEF/DSL package:

```
gencat SpManagedNode.cat SpManagedNode.msg
```

The binary file (SpManagedNode.cat) has to be moved to the $TIVOLIDIR/msg_cat/C directory (note that all directories under msg_cat are just links to the C directory). Make sure that root bin owns the file and that permissions are set to 755. SpManagedNode is now ready to be referenced from any dialog.

In our sp_cws.sp_applications dialog, we have used liberally our message catalog facility. Consider the attributes block for the group of all buttons to launch SP Perspectives views (which was also discussed in section 6.5, “TME 10 Desktop Dialogs” on page 123), as shown in the following example:

```
attributes
          {
           Name = g1;
           Title = Msg(SpManagedNode, "SP Perspectives", 2);
           TitlePos = TOP;
           Border = YES;
           Layout = VERTICAL;
           ChildColumnAlignment = STRETCH;
          }
```

The title attribute is the value of the msg call. This call returns message number 2 of catalog SpManagedNode. In the case of any errors (for example, the message catalog was not found), it also returns the string “SP Perspectives.” That is, “SP Perspectives” is the default message and the message in SpManagedNode is the standard message.

Changing the standard messages of SpManagedNode to a language other than English is easy. After SpManagedNode.msg has been edited, the user should follow the preceding procedure to build and install the new SpManagedNode. Dialogs that reference this catalog do not need to be recompiled.

Note that there is another way to implement message catalogs, this is by using the key facility, where each message is preceded by a string that serves as a key to the message, thus eliminating the need to reference the number of the message in the catalog.

However, using the key facility is more cumbersome and may have disadvantages. It requires not only building the catalog (gencat), but also running a preprocessor (cpp-dsl) to insert in the source dialog file the default messages. This preprocessor only comes with the ADE package, which may not be available in some installations. Furthermore, whenever the dialog is extracted (by using wgetdialog) the preprocessed source file is what is obtained, not the original source file. That is, the unprocessed file is lost for future customizations.

For these reasons, we did not use the key facility, but limited ourselves to referencing the integer keys of the messages. For more information on how to use keyed message catalogs, refer to *TME 10 AEF User's Guide*.

Appendix A. Event Management Resource Variables

This appendix includes a list of all the PSSP 2.2 Event Management resource variables with their data types. It also contains a list of all the Event Management rexource class definitions and the default resource monitors provided with PSSP 2.2.

A.1.1 A List of All Resource Variables

Resource Variable Type	Resource Variable Data Type	Resource Variable Name
Quantity	long	IBM.PSSP.CSS.bcast_rx_ok
Quantity	long	IBM.PSSP.CSS.bcast_tx_ok
Quantity	long	IBM.PSSP.CSS.ibadpackets
Quantity	long	IBM.PSSP.CSS.ibytes_dlt
Quantity	long	IBM.PSSP.CSS.ibytes_lsw
Quantity	long	IBM.PSSP.CSS.ibytes_msw
Quantity	long	IBM.PSSP.CSS.ierrors
Quantity	long	IBM.PSSP.CSS.ipackets_dlt
Quantity	long	IBM.PSSP.CSS.ipackets_drop
Quantity	long	IBM.PSSP.CSS.ipackets_lsw
Quantity	long	IBM.PSSP.CSS.ipackets_msw
Quantity	long	IBM.PSSP.CSS.nobufs
Quantity	long	IBM.PSSP.CSS.obytes_dlt
Quantity	long	IBM.PSSP.CSS.obytes_lsw
Quantity	long	IBM.PSSP.CSS.obytes_msw
Quantity	long	IBM.PSSP.CSS.oerrors
Quantity	long	IBM.PSSP.CSS.opackets_dlt
Quantity	long	IBM.PSSP.CSS.opackets_drop
Quantity	long	IBM.PSSP.CSS.opackets_lsw
Quantity	long	IBM.PSSP.CSS.opackets_msw
Quantity	long	IBM.PSSP.CSS.recvintr_dlt
Quantity	long	IBM.PSSP.CSS.recvintr_lsw
Quantity	long	IBM.PSSP.CSS.recvintr_msw
Quantity	long	IBM.PSSP.CSS.xmitintr_dlt
Quantity	long	IBM.PSSP.CSS.xmitintr_lsw
Quantity	long	IBM.PSSP.CSS.xmitintr_msw
Quantity	long	IBM.PSSP.CSS.xmitque_cur
Quantity	long	IBM.PSSP.CSS.xmitque_max
Quantity	long	IBM.PSSP.CSS.xmitque_ovf
Quantity	long	IBM.PSSP.HARMLD.err_count
Quantity	long	IBM.PSSP.HARMLD.mgrs_conn
Quantity	long	IBM.PSSP.HARMLD.refresh_cntr
Quantity	long	IBM.PSSP.LL.SCHEDD.current_jobs
Quantity	long	IBM.PSSP.LL.SCHEDD.failed_connections
Quantity	long	IBM.PSSP.LL.SCHEDD.failed_in_transactions
Quantity	long	IBM.PSSP.LL.SCHEDD.failed_out_transactions
Quantity	long	IBM.PSSP.LL.SCHEDD.jobs_idle
Quantity	long	IBM.PSSP.LL.SCHEDD.jobs_pending
Quantity	long	IBM.PSSP.LL.SCHEDD.jobs_running
Quantity	long	IBM.PSSP.LL.SCHEDD.jobs_starting
Quantity	long	IBM.PSSP.LL.SCHEDD.total_connections
Quantity	long	IBM.PSSP.LL.SCHEDD.total_in_transactions
Quantity	long	IBM.PSSP.LL.SCHEDD.total_jobs_completed

```
Quantity     long          IBM.PSSP.LL.SCHEDD.total_jobs_rejected
Quantity     long          IBM.PSSP.LL.SCHEDD.total_jobs_removed
Quantity     long          IBM.PSSP.LL.SCHEDD.total_jobs_submitted
Quantity     long          IBM.PSSP.LL.SCHEDD.total_jobs_vacated
Quantity     long          IBM.PSSP.LL.SCHEDD.total_out_transactions
Quantity     long          IBM.PSSP.LL.STARTD.current_jobs
Quantity     long          IBM.PSSP.LL.STARTD.failed_connections
Quantity     long          IBM.PSSP.LL.STARTD.failed_in_transactions
Quantity     long          IBM.PSSP.LL.STARTD.failed_out_transactions
Quantity     long          IBM.PSSP.LL.STARTD.jobs_pending
Quantity     long          IBM.PSSP.LL.STARTD.jobs_running
Quantity     long          IBM.PSSP.LL.STARTD.jobs_suspended
Quantity     long          IBM.PSSP.LL.STARTD.total_connections
Quantity     long          IBM.PSSP.LL.STARTD.total_in_transactions
Quantity     long          IBM.PSSP.LL.STARTD.total_jobs_completed
Quantity     long          IBM.PSSP.LL.STARTD.total_jobs_received
Quantity     long          IBM.PSSP.LL.STARTD.total_jobs_rejected
Quantity     long          IBM.PSSP.LL.STARTD.total_jobs_removed
Quantity     long          IBM.PSSP.LL.STARTD.total_jobs_suspended
Quantity     long          IBM.PSSP.LL.STARTD.total_jobs_vacated
Quantity     long          IBM.PSSP.LL.STARTD.total_out_transactions
State        long          IBM.PSSP.Membership.LANAdapter.state
State        long          IBM.PSSP.Membership.Node.state
Quantity     long          IBM.PSSP.PRCRS.procs_online
State        SBS           IBM.PSSP.Prog.pcount
State        SBS           IBM.PSSP.Prog.xpcount
State        long          IBM.PSSP.Response.Host.state
State        long          IBM.PSSP.Response.Switch.state
State        long          IBM.PSSP.SP_HW.Frame.amp_ARange
State        long          IBM.PSSP.SP_HW.Frame.amp_BRange
State        long          IBM.PSSP.SP_HW.Frame.amp_CRange
State        long          IBM.PSSP.SP_HW.Frame.amp_DRange
State        long          IBM.PSSP.SP_HW.Frame.applicationLevel
State        long          IBM.PSSP.SP_HW.Frame.codeVersion
State        long          IBM.PSSP.SP_HW.Frame.controllerIDMismatch
State        long          IBM.PSSP.SP_HW.Frame.controllerResponds
State        long          IBM.PSSP.SP_HW.Frame.controllerTail
State        long          IBM.PSSP.SP_HW.Frame.controllerTail1LED
State        long          IBM.PSSP.SP_HW.Frame.controllerTail2LED
State        long          IBM.PSSP.SP_HW.Frame.controllerTailActive
State        long          IBM.PSSP.SP_HW.Frame.daemonPollRate
State        long          IBM.PSSP.SP_HW.Frame.feepromEmpty
State        long          IBM.PSSP.SP_HW.Frame.feepromEraseCount
State        long          IBM.PSSP.SP_HW.Frame.frACLED
State        long          IBM.PSSP.SP_HW.Frame.frDCLED
State        long          IBM.PSSP.SP_HW.Frame.frNoPowerModA
State        long          IBM.PSSP.SP_HW.Frame.frNoPowerModB
State        long          IBM.PSSP.SP_HW.Frame.frNoPowerModC
State        long          IBM.PSSP.SP_HW.Frame.frNoPowerModD
State        long          IBM.PSSP.SP_HW.Frame.frNodeComm
State        long          IBM.PSSP.SP_HW.Frame.frPowerDC_A
State        long          IBM.PSSP.SP_HW.Frame.frPowerDC_B
State        long          IBM.PSSP.SP_HW.Frame.frPowerDC_C
State        long          IBM.PSSP.SP_HW.Frame.frPowerDC_D
State        long          IBM.PSSP.SP_HW.Frame.frPowerLED
State        long          IBM.PSSP.SP_HW.Frame.frPowerModAbad
State        long          IBM.PSSP.SP_HW.Frame.frPowerModBbad
State        long          IBM.PSSP.SP_HW.Frame.frPowerModCbad
State        long          IBM.PSSP.SP_HW.Frame.frPowerModDbad
```

```
State       long        IBM.PSSP.SP_HW.Frame.frPowerMods
State       long        IBM.PSSP.SP_HW.Frame.frPowerOff
State       long        IBM.PSSP.SP_HW.Frame.frPowerOff_A
State       long        IBM.PSSP.SP_HW.Frame.frPowerOff_B
State       long        IBM.PSSP.SP_HW.Frame.frPowerOff_C
State       long        IBM.PSSP.SP_HW.Frame.frPowerOff_D
State       long        IBM.PSSP.SP_HW.Frame.frRS232Active
State       long        IBM.PSSP.SP_HW.Frame.frRS232Active1
State       long        IBM.PSSP.SP_HW.Frame.frRS232Active2
State       long        IBM.PSSP.SP_HW.Frame.frTTNodeComm
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen1
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen2
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen3
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen4
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen5
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen6
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen7
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen8
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen9
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen10
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen11
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen12
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen13
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen14
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen15
State       long        IBM.PSSP.SP_HW.Frame.nodeLinkOpen16
State       long        IBM.PSSP.SP_HW.Frame.nodefail1
State       long        IBM.PSSP.SP_HW.Frame.nodefail2
State       long        IBM.PSSP.SP_HW.Frame.nodefail3
State       long        IBM.PSSP.SP_HW.Frame.nodefail4
State       long        IBM.PSSP.SP_HW.Frame.nodefail5
State       long        IBM.PSSP.SP_HW.Frame.nodefail6
State       long        IBM.PSSP.SP_HW.Frame.nodefail7
State       long        IBM.PSSP.SP_HW.Frame.nodefail8
State       long        IBM.PSSP.SP_HW.Frame.nodefail9
State       long        IBM.PSSP.SP_HW.Frame.nodefail10
State       long        IBM.PSSP.SP_HW.Frame.nodefail11
State       long        IBM.PSSP.SP_HW.Frame.nodefail12
State       long        IBM.PSSP.SP_HW.Frame.nodefail13
State       long        IBM.PSSP.SP_HW.Frame.nodefail14
State       long        IBM.PSSP.SP_HW.Frame.nodefail15
State       long        IBM.PSSP.SP_HW.Frame.nodefail16
State       long        IBM.PSSP.SP_HW.Frame.nodefail17
State       long        IBM.PSSP.SP_HW.Frame.rs232CTS
State       long        IBM.PSSP.SP_HW.Frame.rs232DCD
State       long        IBM.PSSP.SP_HW.Frame.serialNum
State       long        IBM.PSSP.SP_HW.Frame.supFailure
State       long        IBM.PSSP.SP_HW.Frame.tempRange
State       long        IBM.PSSP.SP_HW.Frame.type
State       long        IBM.PSSP.SP_HW.Frame.voltP48Range
State       long        IBM.PSSP.SP_HW.Node.P2_5dPresent
State       long        IBM.PSSP.SP_HW.Node.P48OK
State       long        IBM.PSSP.SP_HW.Node.P48Off10Sec
State       long        IBM.PSSP.SP_HW.Node.P4dPresent
State       long        IBM.PSSP.SP_HW.Node.P5DCok
State       long        IBM.PSSP.SP_HW.Node.codeVersion
State       long        IBM.PSSP.SP_HW.Node.currentShutdown
State       long        IBM.PSSP.SP_HW.Node.envLED
State       long        IBM.PSSP.SP_HW.Node.fanfail1
```

```
State       long        IBM.PSSP.SP_HW.Node.fanfail1d
State       long        IBM.PSSP.SP_HW.Node.fanfail2
State       long        IBM.PSSP.SP_HW.Node.fanfail2d
State       long        IBM.PSSP.SP_HW.Node.fanfail3
State       long        IBM.PSSP.SP_HW.Node.fanfail3d
State       long        IBM.PSSP.SP_HW.Node.fanfail4
State       long        IBM.PSSP.SP_HW.Node.fanfail4d
State       long        IBM.PSSP.SP_HW.Node.fanfail5d
State       long        IBM.PSSP.SP_HW.Node.keyModeSwitch
State       SBS         IBM.PSSP.SP_HW.Node.lcd1
State       long        IBM.PSSP.SP_HW.Node.lcd1flash
State       SBS         IBM.PSSP.SP_HW.Node.lcd2
State       long        IBM.PSSP.SP_HW.Node.lcd2flash
State       long        IBM.PSSP.SP_HW.Node.memoryProtect
State       long        IBM.PSSP.SP_HW.Node.nodePower
State       long        IBM.PSSP.SP_HW.Node.nodePowerOn10Sec
State       long        IBM.PSSP.SP_HW.Node.powerLED
State       long        IBM.PSSP.SP_HW.Node.s1PortDTR
State       long        IBM.PSSP.SP_HW.Node.serialLinkOpen
State       long        IBM.PSSP.SP_HW.Node.serialNum
State       long        IBM.PSSP.SP_HW.Node.shutdownN12High
State       long        IBM.PSSP.SP_HW.Node.shutdownN12Low
State       long        IBM.PSSP.SP_HW.Node.shutdownP12High
State       long        IBM.PSSP.SP_HW.Node.shutdownP12Low
State       long        IBM.PSSP.SP_HW.Node.shutdownP2_5dHigh
State       long        IBM.PSSP.SP_HW.Node.shutdownP2_5dLow
State       long        IBM.PSSP.SP_HW.Node.shutdownP48Low
State       long        IBM.PSSP.SP_HW.Node.shutdownP4High
State       long        IBM.PSSP.SP_HW.Node.shutdownP4Low
State       long        IBM.PSSP.SP_HW.Node.shutdownP4dHigh
State       long        IBM.PSSP.SP_HW.Node.shutdownP4dLow
State       long        IBM.PSSP.SP_HW.Node.shutdownP5High
State       long        IBM.PSSP.SP_HW.Node.shutdownP5Low
State       long        IBM.PSSP.SP_HW.Node.shutdownP5iHigh
State       long        IBM.PSSP.SP_HW.Node.shutdownP5iLow
State       long        IBM.PSSP.SP_HW.Node.shutdownP5mHigh
State       long        IBM.PSSP.SP_HW.Node.shutdownP5mLow
State       long        IBM.PSSP.SP_HW.Node.shutdownTemp
State       long        IBM.PSSP.SP_HW.Node.smpDiagLEDoff
State       long        IBM.PSSP.SP_HW.Node.smpPowerLEDoff
State       long        IBM.PSSP.SP_HW.Node.tempRange
State       long        IBM.PSSP.SP_HW.Node.type
State       long        IBM.PSSP.SP_HW.Node.voltN12Range
State       long        IBM.PSSP.SP_HW.Node.voltP12Range
State       long        IBM.PSSP.SP_HW.Node.voltP2_5dRange
State       long        IBM.PSSP.SP_HW.Node.voltP48Range
State       long        IBM.PSSP.SP_HW.Node.voltP4Range
State       long        IBM.PSSP.SP_HW.Node.voltP4dRange
State       long        IBM.PSSP.SP_HW.Node.voltP5Range
State       long        IBM.PSSP.SP_HW.Node.voltP5iRange
State       long        IBM.PSSP.SP_HW.Node.voltP5mRange
State       long        IBM.PSSP.SP_HW.Switch.P48OK
State       long        IBM.PSSP.SP_HW.Switch.P48Off10Sec
State       long        IBM.PSSP.SP_HW.Switch.applicationLevel
State       long        IBM.PSSP.SP_HW.Switch.chip
State       long        IBM.PSSP.SP_HW.Switch.chipClkMissing
State       long        IBM.PSSP.SP_HW.Switch.chipReceivedInit
State       long        IBM.PSSP.SP_HW.Switch.chipSelftest
State       long        IBM.PSSP.SP_HW.Switch.clockSource
```

```
State       long        IBM.PSSP.SP_HW.Switch.codeVersion
State       long        IBM.PSSP.SP_HW.Switch.envLED
State       long        IBM.PSSP.SP_HW.Switch.epromErase
State       long        IBM.PSSP.SP_HW.Switch.epromProgram
State       long        IBM.PSSP.SP_HW.Switch.fanfail1
State       long        IBM.PSSP.SP_HW.Switch.fanfail2
State       long        IBM.PSSP.SP_HW.Switch.fanfail3
State       long        IBM.PSSP.SP_HW.Switch.fanfail4
State       long        IBM.PSSP.SP_HW.Switch.fanfail5
State       long        IBM.PSSP.SP_HW.Switch.feepromEmpty
State       long        IBM.PSSP.SP_HW.Switch.feepromEraseCount
State       long        IBM.PSSP.SP_HW.Switch.mux
State       long        IBM.PSSP.SP_HW.Switch.nodePower
State       long        IBM.PSSP.SP_HW.Switch.nodePowerOn10Sec
State       long        IBM.PSSP.SP_HW.Switch.osc
State       long        IBM.PSSP.SP_HW.Switch.pll
State       long        IBM.PSSP.SP_HW.Switch.port
State       long        IBM.PSSP.SP_HW.Switch.portClkMissing
State       long        IBM.PSSP.SP_HW.Switch.powerLED
State       long        IBM.PSSP.SP_HW.Switch.powerOnReset
State       long        IBM.PSSP.SP_HW.Switch.powerS1
State       long        IBM.PSSP.SP_HW.Switch.powerS2
State       long        IBM.PSSP.SP_HW.Switch.ps1Fail
State       long        IBM.PSSP.SP_HW.Switch.ps1FuseGoodRange
State       long        IBM.PSSP.SP_HW.Switch.ps1PowerGoodRange
State       long        IBM.PSSP.SP_HW.Switch.ps2Fail
State       long        IBM.PSSP.SP_HW.Switch.ps2FuseGoodRange
State       long        IBM.PSSP.SP_HW.Switch.ps2PowerGoodRange
State       long        IBM.PSSP.SP_HW.Switch.psParallelFail
State       long        IBM.PSSP.SP_HW.Switch.psParallelRange
State       long        IBM.PSSP.SP_HW.Switch.recPortNotTuned
State       long        IBM.PSSP.SP_HW.Switch.sendPortNotTuned
State       long        IBM.PSSP.SP_HW.Switch.serialNum
State       long        IBM.PSSP.SP_HW.Switch.shutdownN5High
State       long        IBM.PSSP.SP_HW.Switch.shutdownN5Low
State       long        IBM.PSSP.SP_HW.Switch.shutdownOC
State       long        IBM.PSSP.SP_HW.Switch.shutdownP12High
State       long        IBM.PSSP.SP_HW.Switch.shutdownP12Low
State       long        IBM.PSSP.SP_HW.Switch.shutdownP3_3High
State       long        IBM.PSSP.SP_HW.Switch.shutdownP3_3Low
State       long        IBM.PSSP.SP_HW.Switch.shutdownP48Low
State       long        IBM.PSSP.SP_HW.Switch.shutdownP5High
State       long        IBM.PSSP.SP_HW.Switch.shutdownP5Low
State       long        IBM.PSSP.SP_HW.Switch.shutdownTemp
State       long        IBM.PSSP.SP_HW.Switch.subtype
State       long        IBM.PSSP.SP_HW.Switch.supFailure
State       long        IBM.PSSP.SP_HW.Switch.synchReset
State       long        IBM.PSSP.SP_HW.Switch.tempRange
State       long        IBM.PSSP.SP_HW.Switch.type
State       long        IBM.PSSP.SP_HW.Switch.voltN5Range
State       long        IBM.PSSP.SP_HW.Switch.voltP12Range
State       long        IBM.PSSP.SP_HW.Switch.voltP3_3Range
State       long        IBM.PSSP.SP_HW.Switch.voltP48Range
State       long        IBM.PSSP.SP_HW.Switch.voltP5Range
Counter     long        IBM.PSSP.VSD.bytes_read
Counter     long        IBM.PSSP.VSD.bytes_write
Counter     long        IBM.PSSP.VSD.cache_hits
Counter     long        IBM.PSSP.VSD.client_req_read
Counter     long        IBM.PSSP.VSD.client_req_write
```

```
Counter     long        IBM.PSSP.VSD.local_req_read
Counter     long        IBM.PSSP.VSD.local_req_write
Counter     long        IBM.PSSP.VSD.physical_req_read
Counter     long        IBM.PSSP.VSD.physical_req_write
Counter     long        IBM.PSSP.VSD.remote_req_read
Counter     long        IBM.PSSP.VSD.remote_req_write
Quantity    long        IBM.PSSP.VSD.server
Quantity    long        IBM.PSSP.VSD.state
Counter     long        IBM.PSSP.VSDdrv.1_retry_count
Counter     long        IBM.PSSP.VSDdrv.2_retry_count
Counter     long        IBM.PSSP.VSDdrv.3_retry_count
Counter     long        IBM.PSSP.VSDdrv.4_retry_count
Counter     long        IBM.PSSP.VSDdrv.5_retry_count
Counter     long        IBM.PSSP.VSDdrv.6_retry_count
Counter     long        IBM.PSSP.VSDdrv.7_retry_count
Counter     long        IBM.PSSP.VSDdrv.8_retry_count
Counter     long        IBM.PSSP.VSDdrv.9_retry_count
Quantity    float       IBM.PSSP.VSDdrv.avg_buddy_wait
Counter     long        IBM.PSSP.VSDdrv.buddy_buffer_shortage
Counter     long        IBM.PSSP.VSDdrv.cache_shortage
Counter     long        IBM.PSSP.VSDdrv.comm_buf_shortage
Counter     long        IBM.PSSP.VSDdrv.indirect_io
Counter     long        IBM.PSSP.VSDdrv.pbuf_shortage
Counter     long        IBM.PSSP.VSDdrv.rejected_no_buddy_buffer
Counter     long        IBM.PSSP.VSDdrv.rejected_requests
Counter     long        IBM.PSSP.VSDdrv.rejected_responds
Counter     long        IBM.PSSP.VSDdrv.request_block_shortage
Counter     long        IBM.PSSP.VSDdrv.request_rework
Counter     long        IBM.PSSP.VSDdrv.timeout_error
Quantity    float       IBM.PSSP.aixos.CPU.glidle
Quantity    float       IBM.PSSP.aixos.CPU.glkern
Quantity    float       IBM.PSSP.aixos.CPU.gluser
Quantity    float       IBM.PSSP.aixos.CPU.glwait
Counter     long        IBM.PSSP.aixos.Disk.busy
Counter     long        IBM.PSSP.aixos.Disk.rblk
Counter     long        IBM.PSSP.aixos.Disk.wblk
Counter     long        IBM.PSSP.aixos.Disk.xfer
Quantity    float       IBM.PSSP.aixos.FS.%nodesused
Quantity    float       IBM.PSSP.aixos.FS.%totused
Counter     long        IBM.PSSP.aixos.LAN.rcverrors
Counter     long        IBM.PSSP.aixos.LAN.recvdrops
Counter     long        IBM.PSSP.aixos.LAN.xmitdrops
Counter     long        IBM.PSSP.aixos.LAN.xmiterrors
Counter     long        IBM.PSSP.aixos.LAN.xmitovfl
Counter     long        IBM.PSSP.aixos.Mem.Kmem.calls
Counter     long        IBM.PSSP.aixos.Mem.Kmem.failures
Quantity    long        IBM.PSSP.aixos.Mem.Kmem.inuse
Quantity    long        IBM.PSSP.aixos.Mem.Kmem.memuse
Quantity    long        IBM.PSSP.aixos.Mem.Real.%free
Quantity    long        IBM.PSSP.aixos.Mem.Real.%pinned
Quantity    long        IBM.PSSP.aixos.Mem.Real.numfrb
Quantity    long        IBM.PSSP.aixos.Mem.Real.size
Counter     long        IBM.PSSP.aixos.Mem.Virt.pagein
Counter     long        IBM.PSSP.aixos.Mem.Virt.pageout
Counter     long        IBM.PSSP.aixos.Mem.Virt.pagexct
Counter     long        IBM.PSSP.aixos.Mem.Virt.pgspgin
Counter     long        IBM.PSSP.aixos.Mem.Virt.pgspgout
Quantity    float       IBM.PSSP.aixos.PagSp.%totalfree
Quantity    float       IBM.PSSP.aixos.PagSp.%totalused
```

Quantity	long	IBM.PSSP.aixos.PagSp.totalfree
Quantity	long	IBM.PSSP.aixos.PagSp.totalsize
Quantity	float	IBM.PSSP.aixos.Proc.runque
Quantity	float	IBM.PSSP.aixos.Proc.swpque
Quantity	long	IBM.PSSP.aixos.VG.free
Quantity	float	IBM.PSSP.aixos.cpu.idle
Quantity	float	IBM.PSSP.aixos.cpu.kern
Quantity	float	IBM.PSSP.aixos.cpu.user
Quantity	float	IBM.PSSP.aixos.cpu.wait
Quantity	long	IBM.PSSP.aixos.pagsp.%free
Quantity	long	IBM.PSSP.aixos.pagsp.size
State	SBS	IBM.PSSP.pm.Errlog
State	SBS	IBM.PSSP.pm.User_state1
State	SBS	IBM.PSSP.pm.User_state2
State	SBS	IBM.PSSP.pm.User_state3
State	SBS	IBM.PSSP.pm.User_state4
State	SBS	IBM.PSSP.pm.User_state5
State	SBS	IBM.PSSP.pm.User_state6
State	SBS	IBM.PSSP.pm.User_state7
State	SBS	IBM.PSSP.pm.User_state8
State	SBS	IBM.PSSP.pm.User_state9
State	SBS	IBM.PSSP.pm.User_state10
State	SBS	IBM.PSSP.pm.User_state11
State	SBS	IBM.PSSP.pm.User_state12
State	SBS	IBM.PSSP.pm.User_state13
State	SBS	IBM.PSSP.pm.User_state14
State	SBS	IBM.PSSP.pm.User_state15
State	SBS	IBM.PSSP.pm.User_state16

A.1.2 The Resource Class Definitions

Resource Variable Class	Resource_monitor Name	Observation Interval	Reporting Interval
IBM.PSSP.CSS	IBM.PSSP.harmld	5	5
IBM.PSSP.HARMLD	IBM.PSSP.harmld	30	30
IBM.PSSP.LL	IBM.PSSP.harmld	250	250
IBM.PSSP.Membership	Membership	0	0
IBM.PSSP.PRCRS	IBM.PSSP.harmld	86400	86400
IBM.PSSP.Prog	IBM.PSSP.harmpd	0	0
IBM.PSSP.Response	Response	0	0
IBM.PSSP.SP_HW	IBM.PSSP.hmrmd	0	0
IBM.PSSP.VSD	IBM.PSSP.harmld	60	10
IBM.PSSP.aixos.CPU	aixos	15	0
IBM.PSSP.aixos.Disk	aixos	30	0
IBM.PSSP.aixos.FS	aixos	60	0
IBM.PSSP.aixos.LAN	aixos	40	0
IBM.PSSP.aixos.Mem	aixos	15	0
IBM.PSSP.aixos.PagSp	aixos	30	0
IBM.PSSP.aixos.Proc	aixos	60	0
IBM.PSSP.pm	IBM.PSSP.pmanrmd	0	0

A.1.3 The Default Resource Monitors

Resource Monitor Name	Resource Monitor Path	Resource Monitor Arguments
IBM.PSSP.harmld	/usr/lpp/ssp/bin/haemRM/harmld	-f
IBM.PSSP.harmpd	/usr/lpp/ssp/bin/haemRM/harmpd	
IBM.PSSP.hmrmd	/usr/lpp/ssp/bin/haemRM/hmrmd	IBM.PSSP.hmrmd
IBM.PSSP.pmanrmd	/usr/lpp/ssp/bin/pmand	
Membership	(internal)	
Response	(internal)	
aixos	(internal)	

Appendix B. The SP MIBs

This appendix lists the SP MIBs distributed with PSSP 2.2.

The following list was extracted from the /etc/mib.defs file:

```
-- object definitions compiled from RFC1155-SMI { iso 3 6 1  }

internet             iso.3.6.1
directory            internet.1
mgmt                 internet.2
experimental         internet.3
private              internet.4
enterprises          private.1

-- object definitions compiled from IBMSP-MIB { iso 3 6 1  }

ibm                     enterprises.2
ibmProd                 ibm.6
ibmSP                   ibmProd.117
ibmSPConfig             ibmSP.1
ibmSPEMVariables        ibmSP.3

ibmSPhostnodenumber                ibmSPConfig.1             INTEGER
ibmSPhostpartaddr                  ibmSPConfig.2             IpAddress
ibmSPCWScodeversion                ibmSPConfig.3             DisplayString
ibmSPprimaryCWSname                ibmSPConfig.4             DisplayString
ibmSPprimaryCWSoperstatus          ibmSPConfig.5             INTEGER
ibmSPbackupCWSname                 ibmSPConfig.6             DisplayString
ibmSPbackupCWSoperstatus           ibmSPConfig.7             INTEGER
ibmSPSystemTable                   ibmSPConfig.8             Aggregate
ibmSPNodeEntry                     ibmSPSystemTable.1        Aggregate
ibmSPpartitionaddr                 ibmSPNodeEntry.1          IpAddress
ibmSPnodenumber                    ibmSPNodeEntry.2          INTEGER
ibmSPframenumber                   ibmSPNodeEntry.3          INTEGER
ibmSPslotnumber                    ibmSPNodeEntry.4          INTEGER
ibmSPslotsused                     ibmSPNodeEntry.5          INTEGER
ibmSPinitialhostname               ibmSPNodeEntry.6          DisplayString
ibmSPreliablehostname              ibmSPNodeEntry.7          DisplayString
ibmSPsysparname                    ibmSPNodeEntry.8          DisplayString
ibmSPcodeversion                   ibmSPNodeEntry.9          DisplayString
ibmSPErrlogVars                    ibmSP.2                   Aggregate
ibmSPellabel                       ibmSPErrlogVars.1         DisplayString
ibmSPelidentifier                  ibmSPErrlogVars.2         DisplayString
ibmSPeldatetime                    ibmSPErrlogVars.3         DisplayString
ibmSPelsequencenum                 ibmSPErrlogVars.4         DisplayString
ibmSPelmachineid                   ibmSPErrlogVars.5         DisplayString
ibmSPelnodeid                      ibmSPErrlogVars.6         DisplayString
ibmSPelclass                       ibmSPErrlogVars.7         DisplayString
ibmSPeltype                        ibmSPErrlogVars.8         DisplayString
ibmSPelresource                    ibmSPErrlogVars.9         DisplayString
ibmSPelrscclass                    ibmSPErrlogVars.10        DisplayString
ibmSPelrsctype                     ibmSPErrlogVars.11        DisplayString
ibmSPellocation                    ibmSPErrlogVars.12        DisplayString
ibmSPelvpd                         ibmSPErrlogVars.13        DisplayString
ibmSPetdescription                 ibmSPErrlogVars.14        DisplayString
ibmSPetprobcauses                  ibmSPErrlogVars.15        DisplayString
```

ibmSPetusercauses	ibmSPErrlogVars.16	DisplayString
ibmSPetuseraction	ibmSPErrlogVars.17	DisplayString
ibmSPetinstcauses	ibmSPErrlogVars.18	DisplayString
ibmSPetinstaction	ibmSPErrlogVars.19	DisplayString
ibmSPetfailcauses	ibmSPErrlogVars.20	DisplayString
ibmSPetfailaction	ibmSPErrlogVars.21	DisplayString
ibmSPeldetaildata	ibmSPErrlogVars.22	DisplayString
ibmSPEMEvent	ibmSPEMVariables.1	Aggregate
ibmSPEMEventID	ibmSPEMEvent.1	INTEGER
ibmSPEMEventFlags	ibmSPEMEvent.2	INTEGER
ibmSPEMEventTime	ibmSPEMEvent.3	TimeTicks
ibmSPEMEventLocation	ibmSPEMEvent.4	INTEGER
ibmSPEMEventPartitionAddress	ibmSPEMEvent.5	IpAddress
ibmSPEMEventVarsTableName	ibmSPEMEvent.6	DisplayString
ibmSPEMEventVarsTableInstanceID	ibmSPEMEvent.7	DisplayString
ibmSPEMEventVarName	ibmSPEMEvent.8	DisplayString
ibmSPEMEventVarValueInstanceVector	ibmSPEMEvent.9	DisplayString
ibmSPEMEventVarValuesTableInstanceID	ibmSPEMEvent.10	DisplayString
ibmSPEMEventVarValue	ibmSPEMEvent.11	DisplayString
ibmSPEMEventPredicate	ibmSPEMEvent.12	DisplayString
ibmSPEMNodeDepVarsTable	ibmSPEMVariables.2	Aggregate
ibmSPEMNodeDepVarEntry	ibmSPEMNodeDepVarsTable.1	Aggregate
ibmSPEMNodeDepVarName	ibmSPEMNodeDepVarEntry.1	DisplayString
ibmSPEMNodeDepVarDescr	ibmSPEMNodeDepVarEntry.2	DisplayString
ibmSPEMNodeDepVarType	ibmSPEMNodeDepVarEntry.3	DisplayString
ibmSPEMNodeDepVarDataType	ibmSPEMNodeDepVarEntry.4	DisplayString
ibmSPEMNodeDepVarSBSFormat	ibmSPEMNodeDepVarEntry.5	DisplayString
ibmSPEMNodeDepVarInitValue	ibmSPEMNodeDepVarEntry.6	DisplayString
ibmSPEMNodeDepVarCurValueIndex	ibmSPEMNodeDepVarEntry.7	INTEGER
ibmSPEMNodeDepVarClass	ibmSPEMNodeDepVarEntry.8	DisplayString
ibmSPEMNodeDepVarVecElDefn	ibmSPEMNodeDepVarEntry.9	DisplayString
ibmSPEMNodeDepVarVecElDescr	ibmSPEMNodeDepVarEntry.10	DisplayString
ibmSPEMNodeDepVarPTXName	ibmSPEMNodeDepVarEntry.11	DisplayString
ibmSPEMNodeDepVarDefPred	ibmSPEMNodeDepVarEntry.12	DisplayString
ibmSPEMNodeDepVarEventDescr	ibmSPEMNodeDepVarEntry.13	DisplayString
ibmSPEMNodeDepVarLocator	ibmSPEMNodeDepVarEntry.14	DisplayString
ibmSPEMNodeDepVarOrderGroup	ibmSPEMNodeDepVarEntry.15	DisplayString
ibmSPEMNodeIndepVarsTable	ibmSPEMVariables.3	Aggregate
ibmSPEMNodeIndepVarEntry	ibmSPEMNodeIndepVarsTable.1	Aggregate
ibmSPEMNodeIndepPartaddr	ibmSPEMNodeIndepVarEntry.1	IpAddress
ibmSPEMNodeIndepVarName	ibmSPEMNodeIndepVarEntry.2	DisplayString
ibmSPEMNodeIndepVarDescr	ibmSPEMNodeIndepVarEntry.3	DisplayString
ibmSPEMNodeIndepVarType	ibmSPEMNodeIndepVarEntry.4	DisplayString
ibmSPEMNodeIndepVarDataType	ibmSPEMNodeIndepVarEntry.5	DisplayString
ibmSPEMNodeIndepVarSBSFormat	ibmSPEMNodeIndepVarEntry.6	DisplayString
ibmSPEMNodeIndepVarInitValue	ibmSPEMNodeIndepVarEntry.7	DisplayString
ibmSPEMNodeIndepVarCurValueIndex	ibmSPEMNodeIndepVarEntry.8	INTEGER
ibmSPEMNodeIndepVarClass	ibmSPEMNodeIndepVarEntry.9	DisplayString
ibmSPEMNodeIndepVarVecElDefn	ibmSPEMNodeIndepVarEntry.10	DisplayString
ibmSPEMNodeIndepVarVecElDescr	ibmSPEMNodeIndepVarEntry.11	DisplayString
ibmSPEMNodeIndepVarPTXName	ibmSPEMNodeIndepVarEntry.12	DisplayString
ibmSPEMNodeIndepVarDefPred	ibmSPEMNodeIndepVarEntry.13	DisplayString
ibmSPEMNodeIndepVarEventDescr	ibmSPEMNodeIndepVarEntry.14	DisplayString
ibmSPEMNodeIndepVarOrderGroup	ibmSPEMNodeIndepVarEntry.15	DisplayString
ibmSPEMVarValuesTable	ibmSPEMVariables.4	Aggregate
ibmSPEMVarValuesEntry	ibmSPEMVarValuesTable.1	Aggregate
ibmSPEMVarValueIndex	ibmSPEMVarValuesEntry.1	INTEGER
ibmSPEMVarValueInstanceVector	ibmSPEMVarValuesEntry.2	DisplayString
ibmSPEMVarValuePartaddr	ibmSPEMVarValuesEntry.3	IpAddress

ibmSPEMVarValueName	ibmSPEMVarValuesEntry.4	DisplayString
ibmSPEMVarValue	ibmSPEMVarValuesEntry.5	DisplayString

Appendix C. Contents of the Attached Diskette

This appendix contains a listing of the contents of the diskette distributed with this redbook.

This is a copy of the README file that has been placed on the diskette:

```
(C) Copyright IBM Corp. 1997
All rights reserved.


This diskette is distributed with the IBM publication
  "Integrating TME 10 on the RS/6000 SP", SG24-2071-00

A copy of the files provided on this diskette is also available on the
World Wide Web at the following URL:
   ftp://www.redbooks.ibm.com/redbooks/SG242071
The latest version of these files can be found at that location.

This diskette contains 4 files:
    README -- This file
    tecad_pssp.tar -- A tar file containing all of the necessary files to
       install and execute the PSSP T/EC Adapter.
    SPCustomizations.tar -- A tar file containing all of the necessary files
       to install and customize your TME 10 Desktop with SP-specific
       customizations.
    tasklibs.tar -- A tar file containing 2 sample task libraries to be
       loaded into your TME 10 Framework to perform some common RS/6000 SP
       administrative functions.

For details on using each set of files, refer to the "Integrating
Tivoli on the RS/6000 SP" redbook.  Each tar file contains its own README
with instructions on loading and using the files.

To view the table of contents for one of the tar files, issue the
following AIX command:
   /usr/bin/tar -tvf tar_file
where "tar_file" is the name of the tar file of interest.

To extract the files from one of the tar files, do the following:
1.  Create a working directory to contain the files.
2.  Make the working directory your current directory.
3.  Copy the tar file from the diskette to the current directory.
4.  Extract the files with the following command:
    /usr/bin/tar -xvf tar_file .


The following is the table of contents for the tecad_pssp.tar file:
-rw-r----- 18018 1     7704 Jun 16 10:16:44 1997 README
-rwxr----- 18018 1     1253 Jun 16 10:16:44 1997 install_agent
-rwxr----- 18018 1      590 Jun 16 10:16:45 1997 makeit
-rw-r----- 18018 1     2532 Jun 16 10:16:44 1997 pssp_classes.baroc
-rw-r----- 18018 1    16374 Jun 16 10:16:44 1997 rvclasses.cfg
-rwxr-xr-x 18018 1   130265 Jun 16 13:23:20 1997 tecad_pssp
-rw-r----- 18018 1    28107 Jun 16 10:16:45 1997 tecad_pssp.c
-rw-r----- 18018 1       39 Jun 16 10:16:41 1997 tecad_pssp.cfg
-rwxr----- 18018 1      511 Jun 16 13:22:31 1997 test_agent


The following is the table of contents for the tasklibs.tar file:
-rw------- 19768 1      971 Jun 17 13:50:22 1997 README
-rw-r----- 19768 1     6116 Jun 02 07:51:23 1997 SPTasks.tll
-rw-r----- 19768 1     6556 Jun 02 07:51:37 1997 SwitchTasks.tll


The following is the table of contents for the SPCustomizations.tar file:
drwx--s--- 204 1        0 Jun 03 16:51:02 1997 Resource/
-rw------- 204 1      267 Jun 03 16:23:38 1997 Resource/Makefile
drwxr-sr-x 204 1        0 May 15 00:19:36 1997 Resource/Makefiles/
-rw------- 204 1      266 Feb 25 14:12:58 1997 Resource/Makefiles/Makefile.Bitmap
-rw------- 204 1      347 Feb 25 14:13:03 1997 Resource/Makefiles/Makefile.Callback
-rw------- 204 1      328 Mar 18 09:44:52 1997 Resource/Makefiles/Makefile.Common
-rw------- 204 1      266 Feb 25 14:13:18 1997 Resource/Makefiles/Makefile.Dialog
-rw------- 204 1      248 Feb 21 14:13:32 1997 Resource/Makefiles/Makefile.Icon
-rw------- 204 1      271 Jun 03 16:53:12 1997 Resource/Makefiles/Makefile.MsgCat
-rw------- 204 1      251 May 14 21:53:20 1997 Resource/Makefiles/Makefile.State
-rw------- 204 1      120 May 15 00:13:55 1997 Resource/README
drwxr-sr-x 204 1        0 Jun 03 17:10:10 1997 Resource/ManagedNode/
-rw-r--r--   0 1     1274 Jun 03 17:10:10 1997 Resource/ManagedNode/Makefile
drwxr-sr-x 204 1        0 Jun 03 17:11:31 1997 Resource/ManagedNode/Bitmap/
-rw-r--r--   0 0     4101 May 22 14:41:18 1997 Resource/ManagedNode/Bitmap/sp_cws.xpm
-rw-r--r--   0 0     1701 May 22 14:41:18 1997 Resource/ManagedNode/Bitmap/sp_node.xpm
-rw------- 204 1      476 Feb 25 13:26:32 1997 Resource/ManagedNode/Bitmap/Makefile
-rw------- 204 1     2827 Feb 25 13:26:33 1997 Resource/ManagedNode/Bitmap/sp_cws.xpm.non_perspectives
-rw------- 204 1     2828 Feb 25 13:26:33 1997 Resource/ManagedNode/Bitmap/sp_node.xpm.non_perspectives
drwxr-sr-x 204 1        0 Jun 03 17:12:03 1997 Resource/ManagedNode/Callback/
-rw------- 204 1     1305 May 29 11:02:07 1997 Resource/ManagedNode/Callback/sp_cws.get_all_cw_attributes.ksh
-rw------- 204 1      919 May 29 12:22:16 1997 Resource/ManagedNode/Callback/sp_cws.get_node_numbers.ksh
-rw------- 204 1      566 May 29 12:22:31 1997 Resource/ManagedNode/Callback/sp_cws.launch_applications_driver.ksh
-rw------- 204 1      855 May 29 11:13:31 1997 Resource/ManagedNode/Callback/sp_cws.launch_perspectives.ksh
```

```
-rw------- 204 1      856 May 29 12:22:51 1997 Resource/ManagedNode/Callback/sp_cws.modify_attribute.ksh
-rw------- 204 1      795 May 29 13:03:15 1997 Resource/ManagedNode/Callback/sp_cws.modify_attribute_driver.ksh
-rw------- 204 1     1306 May 29 11:21:56 1997 Resource/ManagedNode/Callback/sp_cws.power_nodes_off.ksh
-rw------- 204 1     1301 May 29 11:22:14 1997 Resource/ManagedNode/Callback/sp_cws.power_nodes_on.ksh
-rw------- 204 1      457 May 29 11:30:54 1997 Resource/ManagedNode/Callback/sp_cws.spmon.ksh
-rw------- 204 1      727 May 29 11:26:01 1997 Resource/ManagedNode/Callback/sp_cws.run_command_driver_nodes.ksh
-rw------- 204 1     1512 May 29 14:58:53 1997 Resource/ManagedNode/Callback/sp_cws.run_command_nodes.ksh
-rw------- 204 1      646 May 29 10:50:03 1997 Resource/ManagedNode/Callback/sp.run_command.ksh
-rw------- 204 1     7251 May 29 15:27:01 1997 Resource/ManagedNode/Callback/Makefile
-rw------- 204 1     1384 May 29 11:34:34 1997 Resource/ManagedNode/Callback/sp_node.get_all_attributes.ksh
-rw------- 204 1      594 May 29 11:38:09 1997 Resource/ManagedNode/Callback/sp_node.get_frame_number.ksh
-rw------- 204 1      512 May 29 11:38:28 1997 Resource/ManagedNode/Callback/sp_node.get_node_number.ksh
-rw------- 204 1      589 May 29 11:38:49 1997 Resource/ManagedNode/Callback/sp_node.get_slot_number.ksh
-rw------- 204 1      825 May 29 11:42:30 1997 Resource/ManagedNode/Callback/sp_node.modify_attribute.ksh
-rw------- 204 1      775 May 29 12:24:43 1997 Resource/ManagedNode/Callback/sp_node.modify_attribute_driver.ksh
-rw------- 204 1      529 May 29 10:47:46 1997 Resource/ManagedNode/Callback/sp.run_command_driver.ksh
-rw------- 204 1     2640 May 29 12:21:36 1997 Resource/ManagedNode/Callback/sp_cws.check_node_response.ksh
-rw------- 204 1      884 May 29 10:58:34 1997 Resource/ManagedNode/Callback/sp_cws.efence_nodes.ksh
-rw------- 204 1      892 May 29 10:59:01 1997 Resource/ManagedNode/Callback/sp_cws.eunfence_nodes.ksh
drwxr-sr-x 204 1        0 Jun 03 17:12:20 1997 Resource/ManagedNode/Dialog/
-rw------- 204 1     1418 May 29 12:59:44 1997 Resource/ManagedNode/Dialog/sp_cws.cw_attributes_dialog
-rw------- 204 1     1248 May 29 13:05:44 1997 Resource/ManagedNode/Dialog/sp_cws.modify_attribute_dialog
-rw------- 204 1     2628 Mar 13 11:51:04 1997 Resource/ManagedNode/Dialog/Makefile
-rw------- 204 1      780 May 19 13:55:32 1997 Resource/ManagedNode/Dialog/sp_cws.run_dialog_nodes
-rw------- 204 1     2627 May 15 11:45:14 1997 Resource/ManagedNode/Dialog/sp_cws.sp_applications
-rw------- 204 1     1257 May 19 13:58:21 1997 Resource/ManagedNode/Dialog/sp_node.modify_attribute_dialog
-rw------- 204 1     1271 May 19 13:59:21 1997 Resource/ManagedNode/Dialog/sp_node.node_attributes_dialog
-rw------- 204 1      717 May 19 13:29:22 1997 Resource/ManagedNode/Dialog/sp.run_dialog
-rw------- 204 1      561 May 19 15:41:29 1997 Resource/ManagedNode/Dialog/sp_cws.node_response
-rw------- 204 1     5732 May 19 13:53:34 1997 Resource/ManagedNode/Dialog/sp_cws.parent_dialog.custom
-rw------- 204 1     4443 May 19 14:08:29 1997 Resource/ManagedNode/Dialog/sp_node.parent_dialog.custom
-rw------- 204 1    12780 May 19 13:54:19 1997 Resource/ManagedNode/Dialog/sp_cws.parent_dialog
-rw------- 204 1    12780 May 19 13:54:19 1997 Resource/ManagedNode/Dialog/sp_cws.parent_dialog.last
-rw------- 204 1    11503 May 19 14:09:00 1997 Resource/ManagedNode/Dialog/sp_node.parent_dialog
-rw------- 204 1    12780 May 19 13:54:19 1997 Resource/ManagedNode/Dialog/sp_cws.parent_dialog.sp
-rw------- 204 1    11503 May 19 14:09:00 1997 Resource/ManagedNode/Dialog/sp_node.parent_dialog.sp
-rw------- 204 1    11503 May 19 14:09:00 1997 Resource/ManagedNode/Dialog/sp_node.parent_dialog.last
drwxr-sr-x 204 1        0 Jun 03 17:12:07 1997 Resource/ManagedNode/Icon/
-rw------- 204 1      604 Feb 25 17:36:46 1997 Resource/ManagedNode/Icon/Makefile
-rw------- 204 1     2738 May 19 14:18:14 1997 Resource/ManagedNode/Icon/sp_cws.custom
-rw------- 204 1     2704 May 19 14:23:58 1997 Resource/ManagedNode/Icon/sp_node.custom
-rw------- 204 1     3448 May 19 14:18:46 1997 Resource/ManagedNode/Icon/sp_cws.last
-rw------- 204 1     3411 May 19 14:24:12 1997 Resource/ManagedNode/Icon/sp_node.last
-rw------- 204 1     3448 May 19 14:18:46 1997 Resource/ManagedNode/Icon/sp_cws
-rw------- 204 1     3411 May 19 14:24:12 1997 Resource/ManagedNode/Icon/sp_node
-rw------- 204 1     3448 May 19 14:18:46 1997 Resource/ManagedNode/Icon/sp_cws.sp
-rw------- 204 1     3411 May 19 14:24:12 1997 Resource/ManagedNode/Icon/sp_node.sp
drwxr-sr-x 204 1        0 Jun 03 17:12:11 1997 Resource/ManagedNode/State/
-rw------- 204 1      491 May 14 21:53:59 1997 Resource/ManagedNode/State/Makefile
-rw------- 204 1        0 Mar 19 12:50:53 1997 Resource/ManagedNode/State/sp_cws
-rw------- 204 1        0 Mar 19 12:50:56 1997 Resource/ManagedNode/State/sp_node
drwxr-sr-x 204 1        0 Mar 26 15:28:51 1997 Resource/ManagedNode/IconExt/
-rw------- 204 1       90 Mar 26 15:28:40 1997 Resource/ManagedNode/IconExt/Makefile
drwxr-sr-x 204 1        0 Jun 03 17:12:08 1997 Resource/ManagedNode/MsgCat/
-rw------- 204 1      196 Jun 03 16:31:32 1997 Resource/ManagedNode/MsgCat/Makefile
-rw------- 204 1     1904 May 19 15:48:52 1997 Resource/ManagedNode/MsgCat/SpManagedNode.msg
-rw-r--r--   0 1     1274 Jun 03 17:08:11 1997 Resource/ManagedNode/Makefile.last
-rw------- 204 1      106 Jun 03 16:53:52 1997 Resource/ManagedNode/LIST
drwxr-sr-x 204 1        0 May 15 00:20:52 1997 Resource/bin/
-rwx------ 204 1      561 Mar 02 21:32:26 1997 Resource/bin/aefInstallIcon
-rwx------ 204 1      416 May 14 22:33:50 1997 Resource/bin/aefUninstallDialog
-rwx------ 204 1      416 Mar 02 21:36:19 1997 Resource/bin/aefUninstallBitmap
-rwx------ 204 1      312 Mar 02 21:39:14 1997 Resource/bin/aefUninstallIcon
-rwx------ 204 1      794 Mar 02 21:37:21 1997 Resource/bin/aefUninstallCallback
-rwx------ 204 1      365 Mar 02 21:33:53 1997 Resource/bin/aefPutRemoteFile
-rwx------ 204 1      222 Mar 19 12:52:53 1997 Resource/bin/aefChangeState
-rwx------ 204 1      442 Mar 02 21:29:01 1997 Resource/bin/aefInstallBitmap
-rwx------ 204 1     1044 Jun 03 16:25:34 1997 Resource/bin/aefInstallCallback
-rwx------ 204 1      649 Mar 19 13:34:44 1997 Resource/bin/aefInstallDialog
-rwx------ 204 1      117 Mar 02 21:34:30 1997 Resource/bin/aefRemoveRemoteFile
-rwx------ 204 1      726 Jun 03 16:55:27 1997 Resource/bin/tivoliPaths
-rwx------ 204 1     5156 Mar 02 21:35:12 1997 Resource/bin/aefResource
-rwx------ 204 1   434681 Nov 07 21:31:04 1996 Resource/bin/gmake
-rwx------ 204 1      543 Feb 25 13:26:10 1997 Resource/bin/aefInstall
-rwx------ 204 1     1175 May 14 22:41:17 1997 Resource/bin/aefUninstall
-rwx------ 204 1     1294 Mar 26 16:01:46 1997 Resource/bin/installAdminManagedNodeIconExtensions
-rwx------ 204 1     1703 May 14 22:55:37 1997 Resource/bin/createManagedNodeMakefile
-rwx------ 204 1      536 Jun 03 17:06:04 1997 Resource/bin/spcustomFastInstall
-rwx------ 204 1     3693 May 14 21:41:39 1997 Resource/bin/spcustomCustomInstall
-rwx------ 204 1       60 May 14 21:47:13 1997 Resource/bin/spcustomUninstall
-rw------- 204 1     2966 Jun 03 14:16:27 1997 Resource/INSTALL
```

Appendix D. Source files for the PSSP T/EC Adapter

The PSSP T/EC Adapter is not a standard part of PSSP and is therefore provided on an as is basis. Support is available through a Programming Request for Price Quotation (PRPQ).

However, in this appendix, we provide you with the source files to adapt the tecad_pssp to your own requirements using the compilation scripts. These files are also included on the diskette attached to this redbook.

The tecad_pssp program is adaptable to new resource classes and resource variables. These classes and variables may come from other programs or resource monitors.

D.1.1 The makeit file

Figure 54 shows a listing of the makeit file which creates the **tecad_pssp** command from the C code tecad_pssp.c.

```
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# add the gcc bin directory to your PATH
export PATH=$PATH:/usr/local/bin

if [[ -z $1 ]]
then
        mode="unsecure"
else
        mode=$1
fi
if [[ $mode = "unsecure" ]]
then
     gcc -g -I/usr/local/Tivoli/include/aix4-r1/tivoli \
         -L/usr/local/Tivoli/lib/aix4-r1 tecad_pssp.c \
         -lteceif -lg++ -lm -o tecad_pssp
exit 0
fi

if [[ $mode = "secure" ]]
then
     gcc -g -I/usr/local/Tivoli/include/aix4-r1/tivoli \
         -L/usr/local/Tivoli/lib/aix4-r1 tecad_pssp.c \
         -ltec -las -las_imp -ltds -lui -ltas -ldes \
         -ltmfimp -ltmf -lthreads -lg++ -lm -o tecad_pssp
exit 0
fi

echo "usage: makeit [unsecure/secure]"
exit 1
```

Figure 54. PSSP T/EC Adapter makeit file

D.1.2 The PSSP T/EC Adapter BAROC File

This section provides a complete listing of the BAROC file tecad_pssp.baroc.

```
# (C) Copyright IBM Corp. 1997
# All rights reserved.

# BASE CLASS FOR ALL PSSP CLASSES #

TEC_CLASS:
      PSSP_EVENT ISA EVENT
      DEFINES
      {
            source: default= "PSSP";
            sub_source: default= "PSSP";
            sub_origin: default= "N/A";
            msg_catalog: default= "none";
            msg_index: default= 0;
            repeat_count: default = 0;
            subscription_handle: STRING;
            predicate: STRING;
            principal: STRING;
            type : STRING;
            resource_variable: STRING;
      };
END

TEC_CLASS:
      PSSP_REAL ISA PSSP_EVENT
      DEFINES
      {
            value: REAL;
            node_number: INTEGER;
      };
END

TEC_CLASS:
      PSSP_CPU ISA PSSP_REAL
      DEFINES
      {
            CPU_number: INTEGER, default=1;
      };
END

TEC_CLASS:
      PSSP_FS ISA PSSP_REAL
      DEFINES
      {
            logical_volume: STRING;
            volume_group: STRING;
      };
END

TEC_CLASS:
      PSSP_LONG ISA PSSP_EVENT
      DEFINES
      {
            value: INT32;
            node_number: INTEGER;
```

```
      };
END

TEC_CLASS:
      PSSP_LLSTART ISA PSSP_LONG
      DEFINES
      {
            STARTD: STRING;
      };
END

TEC_CLASS:
      PSSP_LLSCHED ISA PSSP_LONG
      DEFINES
      {
            SCHEDD: STRING;
      };
END

TEC_CLASS:
      PSSP_PAGESPACE ISA PSSP_LONG
      DEFINES
      {
            name: STRING;
      };
END

TEC_CLASS:
      PSSP_VG ISA PSSP_LONG
      DEFINES
      {
            volume_group: STRING;
      };
END

TEC_CLASS:
      PSSP_KMEM ISA PSSP_LONG
      DEFINES
      {
            memory_type: STRING;
      };
END

TEC_CLASS:
      PSSP_DISK ISA PSSP_LONG
      DEFINES
      {
            disk_name: STRING;
      };
END

TEC_CLASS:
      PSSP_ADAPTER ISA PSSP_LONG
      DEFINES
      {
            adapter_type: STRING;
            adapter_number: INT32;
      };
END
```

```
TEC_CLASS:
      PSSP_PROCESS ISA PSSP_EVENT
      DEFINES
      {
            node_number: INTEGER;
            user_name: STRING;
            program_name: STRING;
            previous_count: INTEGER;
            current_count: INTEGER;
            current_list: STRING;
      };
END

TEC_CLASS:
      PSSP_USER ISA PSSP_EVENT
      DEFINES
      {
            node_number: STRING;
            user_string: STRING;
      };
END

TEC_CLASS:
      PSSP_ERRLOG ISA PSSP_EVENT
      DEFINES
      {
            node_number : INTEGER;
            resource_type: STRING;
            resource_name: STRING;
            alert_flag: STRING;
            error_type: STRING;
            error_class: STRING;
            error_ID: STRING;
            sequence_number: STRING;
            resource_class: STRING;
            error_label: STRING;
      };
END

TEC_CLASS:
      PSSP_VSD ISA PSSP_EVENT
      DEFINES
      {
            L1 : STRING;
            L2 : STRING;
            VSD_name: STRING;
            node_number: INTEGER;
      };
END

TEC_CLASS:
      PSSP_STRING ISA PSSP_EVENT
      DEFINES
      {
            data_string : STRING;
            node_number: INTEGER;
      };
END
```

```
TEC_CLASS:
        PSSP_NETWORK ISA PSSP_LONG
        DEFINES
        {
                adapter: STRING;
        };
END

TEC_CLASS:
      PSSP_SWITCH ISA PSSP_EVENT
      DEFINES
      {
            switch_number: INTEGER;
            value: INT32;
      };
END

TEC_CLASS:
      PSSP_FRAME ISA PSSP_EVENT
      DEFINES
      {
            frame_number: INTEGER;
            value: INT32;
      };
END
```

D.1.3 The tecad_pssp.c File

This section lists the tecad_pssp.c file. You can adapt this file to match your own requirements and compile it using the makeit script listed in Figure 54 on page 153.

```
/*
 * (C) Copyright IBM Corp. 1997
 * All Rights Reserved.
 */

/* test "agent" for either a TME or non-TME connection.  Link with the
 * appropriate libraries to build a TME version or non-TME version
 */

#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <assert.h>
#include <memory.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/stat.h>

#include "agent_comm.h"

#define NAME_SIZE 256
#define PREFIX_SIZE 64
#define NUM_VAR 4
#define NUM_PREFIX 5
```

```
typedef enum
{
  PSSP_LONG,
  PSSP_REAL,
  PSSP_ADAPTER,
  PSSP_PROCESS,
  PSSP_STRING,
  PSSP_NETWORK,
  PSSP_FRAME,
  PSSP_SWITCH,
  PSSP_VSD,
  PSSP_DISK,
  PSSP_FS,
  PSSP_KMEM,
  PSSP_VG,
  PSSP_CPU,
  PSSP_PAGESPACE,
  PSSP_ERRLOG,
  PSSP_LLSTART,
  PSSP_LLSCHED
} class_names_t;



void DoLong(void);
void DoReal(void);
void DoAdapter(void);
void DoProcess(void);
void DoString(void);
void DoNetwork(void);
void DoFrame(void);
void DoSwitch(void);
void DoVSD(void);
void DoDisk(void);
void DoFS(void);
void DoKmem(void);
void DoVG(void);
void DoCPU(void);
void DoPageSpace(void);
void DoErrLog(void);
void DoLLStart(void);
void DoLLSched(void);
void ValidateSeverity(void);

void
print_usage_and_exit(char **argv)
{
    fprintf(stderr, "Usage: agent [-f cfgfile] [-c] [-p port] \
                                        [server_host]\n");
    exit(1);
}

/* Global Variables */
int rc, i, n, theClassNumber;
char *theHandle, *theIVector, *theLocation, *thePredicate,
  *thePrincipal, *theRVCount,*theRVName, *theRVType, *theRVTime;
int theRVCountValue;
char theSeverity[256];
char *theRVArray[NUM_VAR];
```

```
char *theRVPrefix[NUM_PREFIX];
char theClassName[256];
char s[2048], theMessage[256], theAdmin[256];
char tmp_str[256];
char names[18][256];
float real_value =0;
long long_value = 0;

int
main(int argc, char *argv[])
{
    int c;
    extern int     optind;
    extern char    *optarg;
    char           *location = NULL;
    char           cfg_file_name[FILE_LEN];
    char           *cfg_file = NULL;
    unsigned short port = 0;
    tec_handle_t   th;
    extern int     tec_errno;
    int            connectionless = 0;
    int            oneway = 0;
    tec_delivery   type = use_default;
    struct         stat   sbuf;

    FILE *cmdfp;

        /* allocate arrays */
        for( i = 0; i < NUM_VAR; i++ )
           if( (theRVArray[i] = (char *)calloc( NAME_SIZE,sizeof(char)))
                   == NULL)
           {
                   assert(theRVArray[i]);
           }
        for( i = 0; i < NUM_PREFIX; i++)
           if( (theRVPrefix[i] = (char *)calloc( PREFIX_SIZE,sizeof(char)))
                   == NULL)
           {
                   assert(theRVPrefix[i]);
           }

        *theMessage = '\0';
        *theAdmin = '\0';
        *theSeverity = '\0';
        *cfg_file_name= '\0';

        while ((c = getopt(argc, argv, "Ccl:a:m:p:s:")) != -1) {
          switch (c) {
          case 'l':
            /* Specifies location of the agent configuration file */
            strcpy(cfg_file_name, optarg);
            if (stat(cfg_file_name, &sbuf) != 0) {
              /* not existant */
              printf("%s: Not existant\n", cfg_file_name);
              print_usage_and_exit(argv);
            }
            cfg_file = cfg_file_name;
            break;
          case 'p':
```

```
        /* Specifies the port to connect to on the Server */
        if (!isdigit(*optarg)) {
          print_usage_and_exit(argv);
        }
        port = atol(optarg);
        break;
      case 'C':
        oneway++;
      case 'c':
        connectionless++;
        type = connection_less;
        break;
      case 'm':
        strcpy(theMessage, optarg);
        break;
      case 's':
        strcpy(theSeverity, optarg);
        ValidateSeverity();
        break;
      case 'a':
        strcpy(theAdmin,optarg);
        break;
      default:
        print_usage_and_exit(argv);
        break;
      }
    }

    if (optind + 1 == argc) {
      location = argv[optind];
    }

/* If cfg_file is NULL, the agent_init will look for
   /etc/Tivoli/tecad_eif.conf, if this is unavailable, all
   parameters to tec_create_handle must be provided */
    if( *cfg_file_name == '\0')
      {
        if (stat("./tecad_pssp.cfg", &sbuf) != 0)
          {
            /* not existant */
            printf("%s: Not existant\n", "./tecad_pssp.cfg");
            print_usage_and_exit(argv);
          }
        else
          strcpy(cfg_file_name,"./tecad_pssp.cfg");
      }




tecad_pssp_init(cfg_file_name);

if ((th = tec_create_handle(location, port, oneway, type)) == NULL)
  {
    fprintf(stderr, "tec_create_handle failed, errno = %d\n", tec_errno);
    exit(1);
  }

    /*
```

```
 * Try to flush any buffered events before processing new ones.
 * Note: If the TEC Server or Network is down all buffered
 *   events will still try to be sent; there is no abort - yet.
 */
 tec_flush_events(th);

/* Grab all environment variables, and format the event accordingly */

theHandle = getenv( "PMAN_HANDLE" );
theIVector = getenv( "PMAN_IVECTOR" );
theLocation = getenv( "PMAN_LOCATION" );
thePredicate = getenv( "PMAN_PRED" );
thePrincipal = getenv( "PMAN_PRINCIPAL" );
theRVCount = getenv( "PMAN_RVCOUNT" );
theRVCountValue = atoi( theRVCount );
theRVArray[0] = getenv( "PMAN_RVFIELDO" );
if( theRVCountValue >= 2 )
        theRVArray[1] = getenv( "PMAN_RVFIELD1" );
else
        *theRVArray[1] =  '\0';
 if( theRVCountValue == 3 )
        theRVArray[2] = getenv( "PMAN_RVFIELD2" );
else
        *theRVArray[2] =  '\0';
theRVName = getenv( "PMAN_RVNAME" );
theRVType = getenv( "PMAN_RVTYPE" );
theRVTime = getenv( "PMAN_TIME" );  theHandle = getenv( "PMAN_HANDLE" );
theIVector = getenv( "PMAN_IVECTOR" );
theLocation = getenv( "PMAN_LOCATION" );
thePredicate = getenv( "PMAN_PRED" );
thePrincipal = getenv( "PMAN_PRINCIPAL" );
theRVCount = getenv( "PMAN_RVCOUNT" );
theRVCountValue = atoi( theRVCount );
theRVArray[0] = getenv( "PMAN_RVFIELDO" );
if( theRVCountValue >= 2 )
        theRVArray[1] = getenv( "PMAN_RVFIELD1" );
else
        *theRVArray[1] =  '\0';
 if( theRVCountValue == 3 )
        theRVArray[2] = getenv( "PMAN_RVFIELD2" );
else
        *theRVArray[2] =  '\0';
theRVName = getenv( "PMAN_RVNAME" );
theRVType = getenv( "PMAN_RVTYPE" );
theRVTime = getenv( "PMAN_TIME" );

/* We need to classify what type of event to generate */

sscanf(theRVName, \
  "%[^.].%[^.].%[^.].%[^.].%[^.]", \
  theRVPrefix[0],theRVPrefix[1],theRVPrefix[2], \
  theRVPrefix[3],theRVPrefix[4]);


/* initiate the message string */
*s = '\0';

/* The SDRGetObjects command is used to retrieve the class
   type from the SDR */
```

```
sprintf(tmp_str,
  "/usr/lpp/ssp/bin/SDRGetObjects tecad_pssp_Class rv_name==%s class_number",
  theRVName);

if( (cmdfp = popen(tmp_str,"r")) == NULL)
  {
    fprintf(stderr,"The SDRGetObjects command failed\n");
    exit(1);
  }
fscanf(cmdfp,"%s",tmp_str); /* get rid of the header */
if( (n = fscanf(cmdfp,"%d",&theClassNumber)) != 1)
  {
    fprintf(stderr,"The SDRGetObjects command failed\n");
    exit(2);
  }
pclose(cmdfp);

switch( theClassNumber )
{
case PSSP_LONG:
  DoLong();
  break;
case PSSP_REAL:
  DoReal();
  break;
case PSSP_ADAPTER:
  DoAdapter();
  break;
case PSSP_PROCESS:
  DoProcess();
  break;
case PSSP_STRING:
  DoString();
  break;
case PSSP_NETWORK:
  DoNetwork();
  break;
case PSSP_FRAME:
  DoFrame();
  break;
case PSSP_SWITCH:
  DoSwitch();
  break;
case PSSP_VSD:
  DoVSD();
  break;
case PSSP_DISK:
  DoDisk();
  break;
case PSSP_FS:
  DoFS();
  break;
case PSSP_KMEM:
  DoKmem();
  break;
case PSSP_VG:
  DoVG();
  break;
```

```
case PSSP_CPU:
  DoCPU();
  break;
case PSSP_PAGESPACE:
  DoPageSpace();
  break;
case PSSP_ERRLOG:
  DoErrLog();
  break;
case PSSP_LLSTART:
  DoLLStart();
  break;
case PSSP_LLSCHED:
  DoLLSched();
  break;
default:
  fprintf(stderr,
    "tecad_pssp: Could not find class type, using default PSSP_Event\n");
  strcpy(theClassName,"PSSP_EVENT");
  sprintf(s,"%s;source=PSSP;",theClassName);
  break;
}


/* fill in the common slots for PSSP_EVENT */
sprintf(tmp_str,"subscription_handle='%s';",theHandle);
strcat(s,tmp_str);
sprintf(tmp_str,"predicate='%s';",thePredicate);
strcat(s,tmp_str);
sprintf(tmp_str,"principal='%s';",thePrincipal);
strcat(s,tmp_str);
sprintf(tmp_str,"type='%s';",theRVType);
strcat(s,tmp_str);
sprintf(tmp_str,"resource_variable='%s';",theRVName);
strcat(s,tmp_str);
if( *theMessage != '\0')
  {
    sprintf(tmp_str,"msg='%s';",theMessage);
    strcat(s,tmp_str);
  }
if( *theAdmin != '\0')
  {
    sprintf(tmp_str,"administrator='%s';",theAdmin);
    strcat(s,tmp_str);
  }
sprintf(tmp_str,"severity=%s;",theSeverity);
strcat(s,tmp_str);

/* Get the IP of the node number and put it in the origin field */

sscanf(theLocation,"%d",&n);
if( n == 0 )
  {
    sprintf(tmp_str,"host hostname");
  }
 else
  {
    sprintf(tmp_str,"host /usr/lpp/ssp/bin/hostlist -dn %d\n",n);
    if( (cmdfp = popen(tmp_str,"r")) == NULL)
```

```
                {
                  fprintf(stderr,"The hostlist command failed\n");
                  exit(1);
                }
            }
          if( (cmdfp = popen(tmp_str,"r")) == NULL)
            {
              fprintf(stderr,"The hostname command failed\n");
              exit(1);
            }

          fscanf(cmdfp,"%s",tmp_str); /* get the hostname */
          strcat(s,"hostname='");
          strcat(s,tmp_str);
          strcat(s,"';");
          fscanf(cmdfp,"%s",tmp_str); /* get the word 'is' */
          fscanf(cmdfp,"%s",tmp_str); /* get the IP address */
          strcat(s,"origin='");
          strcat(s,tmp_str);
          strcat(s,"';");

          pclose(cmdfp);

          strcat(s,"END\n");

          /* keep the newline and adda ^A to the end, this makes
             the message "<msg>\n\001" */
          s[strlen(s)] = '\001';
          s[strlen(s)+1] = '\0';

          /* tec_put_event will send the event to the selected server */
          rc = tec_put_event(th, s);

          /* you may want to print out the event string to check it
          printf("%s",s);   */

          if (rc == -1) {
            fprintf(stderr, "tec_put_event failed, errno = %d\n", tec_errno);
            exit(1);
          }

          /* eof - close the connection and exit */
          tec_destroy_handle(th);
          exit(0);
}

void DoLong()
{
  strcpy(theClassName,"PSSP_LONG");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_LONG\n");
    }
  else
    {
      strcat(s,"value=");
      strcat(s,theRVArray[0]);
```

```
      strcat(s,";");
    }
  strcat(s,"node_number=");
  strcat(s,theLocation);
  strcat(s,";");
}

void DoReal()
{
  strcpy(theClassName,"PSSP_REAL");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_LONG\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  sprintf(tmp_str,"node_number=%s",theLocation);
  strcat(s,tmp_str);
}

void DoAdapter()
{
  strcpy(theClassName,"PSSP_ADAPTER");
  sprintf(s,"%s;source=PSSP;",theClassName);
  n = sscanf(theIVector, \
      "%[^=]=%[^;];%[^=]=%[^;];%[^=]=%[^;]", \
      names[0],names[1],names[2], \
      names[3],names[4],names[5]);
  if (n != 6 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_ADAPTER");
      exit(1);
    }
  for(i = 0; i < 5; i++)
    {
      if( !strcmp(names[i],"AdapterType") )
        sprintf(tmp_str,"adapter_type=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else if( !strcmp(names[i],"AdapterNum"))
        sprintf(tmp_str,"adapter_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}
void DoProcess()
{
  strcpy(theClassName,"PSSP_PROCESS");
  sprintf(s,"%s;source=PSSP;",theClassName);
  n = sscanf(theIVector, \
      "%[^=]=%[^;];%[^=]=%[^;];%[^=]=%s", \
      names[0],names[1],names[2], \
      names[3],names[4],names[5]);
```

```
  if (n != 6 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_PROCESS");
      exit(1);
    }
  for(i = 0; i < 5; i++)
    {
      if( !strcmp(names[i],"ProgName") )
        sprintf(tmp_str,"program_name=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else if( !strcmp(names[i],"UserName"))
        sprintf(tmp_str,"user_name=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
  if( theRVCountValue != 3)
    {
      fprintf(stderr,"Incorrect number of arguments for the pcount event\n");
      exit(1);
    }
  sscanf(theRVArray[0],"%[^=]=%s",names[0],names[1]);
  sprintf(tmp_str,"current_count=%s;",names[1]);
  strcat(s,tmp_str);
  sscanf(theRVArray[1],"%[^=]=%s",names[0],names[1]);
  sprintf(tmp_str,"previous_count=%s;",names[1]);
  strcat(s,tmp_str);
  sscanf(theRVArray[2],"%[^=]=%s",names[0],names[1]);
  sprintf(tmp_str,"current_list='%s';",names[1]);
  strcat(s,tmp_str);
}

void DoString()
{
  strcpy(theClassName,"PSSP_STRING");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_STRING\n");
    }
  else
    {
      sprintf(tmp_str,"string=%s",theRVArray[0]);
      strcat(s,tmp_str);
    }
  sprintf(tmp_str,"node_number=%s",theLocation);
  strcat(s,tmp_str);
}

void DoNetwork()
{
  strcpy(theClassName,"PSSP_NETWORK");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_NETWORK\n");
```

```
    }
  else
    {
      sprintf(tmp_str,"value=%s",theRVArray[0]);
      strcat(s,tmp_str);
    }
  sprintf(tmp_str,"node_number=%s",theLocation);
  strcat(s,tmp_str);
  n = sscanf(theIVector,"%[^=]=%s",names[0],names[1]);
  if (n != 2 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_Network");
      exit(1);
    }

  if( !strcmp(names[0],"Adapter") )
    {
      sprintf(tmp_str,"adapter=%s;",names[1]);
      strcat(s,tmp_str);
    }
  else
    {
      fprintf(stderr,
        "tecad_pssp: wrong variable in instantiation vector \
                    for variable PSSP_Network\n");
      exit(1);
    }
}

void DoFrame()
{
  strcpy(theClassName,"PSSP_Frame");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_FRAME\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector,"%[^=]=%s",names[0],names[1]);
  if (n != 2 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_Frame");
      exit(1);
    }

  if( !strcmp(names[0],"FrameNum") )
    {
      sprintf(tmp_str,"frame_number=%s;",names[1]);
      strcat(s,tmp_str);
    }
  else
    {
```

```
      fprintf(stderr,
        "tecad_pssp: wrong variable in instantiation vector for variable PSSP_Frame\n");
      exit(1);
    }
}

void DoSwitch()
{
  strcpy(theClassName,"PSSP_Switch");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_SWITCH\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector,"%[^=]=%s",names[0],names[1]);
  if (n != 2 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_Frame");
      exit(1);
    }

  if( !strcmp(names[0],"SwitchNum") )
    {
      sprintf(tmp_str,"switch_number=%s;",names[1]);
      strcat(s,tmp_str);
    }
  else
    {
      fprintf(stderr,
        "tecad_pssp: wrong variable in instantiation vector for variable PSSP_Frame\n");
      exit(1);
    }
}

void DoVSD()
{
  strcpy(theClassName,"PSSP_VSD");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_VSD\n");
    }
    else
      {
        sprintf(tmp_str,"value=%s;",theRVArray[0]);
        strcat(s,tmp_str);
      }
  n = sscanf(theIVector, \
      "%[^=]=%[^;];%[^=]=%[^;]; \
       %[^=]=%[^;];%[^=]=%s", \
       names[0],names[1],names[2],names[3], \
```

```
       names[4],names[5],names[6],names[7]);
  if (n != 8 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_VSD");
      exit(1);
    }
  for(i = 0; i < 5; i++)
    {
      if( !strcmp(names[i],"VSD") )
        sprintf(tmp_str,"VSD_name=%s;",names[i+1]);
      else if( !strcmp(names[i],"L1"))
        sprintf(tmp_str,"L1=%s;",names[i+1]);
      else if( !strcmp(names[i],"L2"))
        sprintf(tmp_str,"L2=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}

void DoDisk()
{
  strcpy(theClassName,"PSSP_Disk");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_Disk\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector, \
      "%[^=]=%[^;];%[^=]=%s", \
       names[0],names[1],names[2],names[3]);
  if (n != 4 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_Disk");
      exit(1);
    }
  for(i = 0; i < 3; i++)
    {
      if( !strcmp(names[i],"Name") )
        sprintf(tmp_str,"disk_name=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}

void DoFS()
{
  strcpy(theClassName,"PSSP_FS");
```

```
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_FS\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector, \
  "%[^=]=%[^;];%[^=]=%[^;];%[^=]=%s", \
  names[0],names[1],names[2],names[3],names[4],names[5]);
  if (n != 6 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_FS");
      exit(1);
    }
  for(i = 0; i < 5; i++)
    {
      if( !strcmp(names[i],"LV") )
        sprintf(tmp_str,"logical_volume=%s;",names[i+1]);
      else if( !strcmp(names[i],"VG"))
        sprintf(tmp_str,"volume_group=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}

void DoKmem()
{
  strcpy(theClassName,"PSSP_Kmem");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_Kmem\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector, \
  "%[^=]=%[^;];%[^=]=%s", \
   names[0],names[1],names[2],names[3]);
  if( n != 4 )
    {
      fprintf(stderr,
        "tecad_pssp: received wrong number of output fields in variable PSSP_KMEM\n");
      exit(1);
    }

  for(i = 0; i < 3; i++)
    {
```

```
      if( !strcmp(names[i],"Type") )
        sprintf(tmp_str,"memory_type=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}


void DoVG()
{
  strcpy(theClassName,"PSSP_VG");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_VG\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector, \
  "%[^=]=%[^;];%[^=]=%s", \
   names[0],names[1],names[2],names[3]);
  if (n != 4 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_VG");
      exit(1);
    }
  for(i = 0; i < 3; i++)
    {
      if( !strcmp(names[i],"VG") )
        sprintf(tmp_str,"volume_group=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}

void DoCPU()
{
  strcpy(theClassName,"PSSP_CPU");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_CPU\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector, \
```

```
  "%[^=]=%[^;];%[^=]=%s", \
   names[0],names[1],names[2],names[3]);
  if (n != 4 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_CPU");
      exit(1);
    }
  for(i = 0; i < 3; i++)
    {
      if( !strcmp(names[i],"CPU") )
        sprintf(tmp_str,"cpu_number=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}

void DoPageSpace()
{
  strcpy(theClassName,"PSSP_PageSpace");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_PageSpace\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
              strcat(s,tmp_str);
    }
  n = sscanf(theIVector, \
  "%[^=]=%[^;];%[^=]=%s", \
   names[0],names[1],names[2],names[3]);
  if (n != 4 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_PageSpace");
      exit(1);
    }
  for(i = 0; i < 3; i++)
    {
      if( !strcmp(names[i],"Name") )
        sprintf(tmp_str,"name=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}

void DoErrLog()
{
  strcpy(theClassName,"PSSP_Errlog");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
```

```
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_Errlog\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector, \
  "%[^=]=%[^;];%[^=]=%[^;];%[^=]=%[^;]; \
   %[^=]=%[^;];%[^=]=%[^;];%[^=]=%[^;]; \
   %[^=]=%[^;];%[^=]=%[^;];%[^=]=%s", \
   names[0],names[1],names[2],names[3], \
   names[4],names[5],names[6],names[7], \
   names[8],names[9],names[10],names[11], \
   names[12],names[13],names[13],names[14], \
   names[15],names[16],names[17]);
  if (n != 18 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_Errlog");
      exit(1);
    }
  for(i = 0; i < 17; i++)
    {
      if( !strcmp(names[i],"resourceType") )
        sprintf(tmp_str,"resource_type=%s;",names[i+1]);
      else if( !strcmp(names[i],"resourceName"))
        sprintf(tmp_str,"resource_name=%s;",names[i+1]);
      else if( !strcmp(names[i],"alertFlagsValue"))
        sprintf(tmp_str,"alert_flags=%s;",names[i+1]);
      else if( !strcmp(names[i],"errorType"))
        sprintf(tmp_str,"error_type=%s;",names[i+1]);
      else if( !strcmp(names[i],"errorClass"))
        sprintf(tmp_str,"errorClass=%s;",names[i+1]);
      else if( !strcmp(names[i],"errorID"))
        sprintf(tmp_str,"error_ID=%s;",names[i+1]);
      else if( !strcmp(names[i],"sequenceNumber"))
        sprintf(tmp_str,"sequence_number=%s;",names[i+1]);
      else if( !strcmp(names[i],"resourceClass"))
        sprintf(tmp_str,"resource_class=%s;",names[i+1]);
      else if( !strcmp(names[i],"errorLabel"))
        sprintf(tmp_str,"error_label=%s;",names[i+1]);
      strcat(s,tmp_str);
    }
}

void DoLLStart()
{
  strcpy(theClassName,"PSSP_STARTD");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_LLStart\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
```

```
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector, \
  "%[^=]=%[^;];%[^=]=%s", \
   names[0],names[1],names[2],names[3]);
  if (n != 4 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_LLStart");
      exit(1);
    }
  for(i = 0; i < 3; i++)
    {
      if( !strcmp(names[i],"STARTD") )
        sprintf(tmp_str,"STARTD=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}

void DoLLSched()
{
  strcpy(theClassName,"PSSP_SCHEDD");
  sprintf(s,"%s;source=PSSP;",theClassName);
  if( theRVCountValue > 1 )
    {
      /* there should not be more than one value here */
      fprintf(stderr,"tec_adapter: wrong argument count for variable PSSP_SCHEDD\n");
    }
  else
    {
      sprintf(tmp_str,"value=%s;",theRVArray[0]);
      strcat(s,tmp_str);
    }
  n = sscanf(theIVector, \
      "%[^=]=%[^;];%[^=]=%s", \
      names[0],names[1],names[2],names[3]);
  if (n != 4 )
    {
      fprintf(stderr,
        "tecad_pssp: the parsing of the IVEC string failed for variable PSSP_SCHEDD");
      exit(1);
    }
  for(i = 0; i < 3; i++)
    {
      if( !strcmp(names[i],"SCHEDD") )
        sprintf(tmp_str,"SCHEDD=%s;",names[i+1]);
      else if( !strcmp(names[i],"NodeNum"))
        sprintf(tmp_str,"node_number=%s;",names[i+1]);
      else continue;
      strcat(s,tmp_str);
    }
}

void ValidateSeverity()
{
  if( strcmp(theSeverity,"FATAL") &&
```

```
      strcmp(theSeverity,"CRITICAL") &&
      strcmp(theSeverity,"WARNING") &&
      strcmp(theSeverity,"MINOR") &&
      strcmp(theSeverity,"HARMLESS") &&
      strcmp(theSeverity,"UNKNOWN"))
      strcpy(theSeverity,"WARNING");

}
```

D.1.4 The rvclasses.cfg file

This is a listing of the **rvclasses.cfg** file. It is used to define the mapping of the Event Management resource variables to T/EC classes and is loaded into the SDR using the install_agent command.

```
IBM.PSSP.CSS.bcast_rx_ok PSSP_LONG 0
IBM.PSSP.CSS.bcast_tx_ok PSSP_LONG 0
IBM.PSSP.CSS.ibadpackets PSSP_LONG 0
IBM.PSSP.CSS.ibytes_dlt PSSP_LONG 0
IBM.PSSP.CSS.ibytes_lsw PSSP_LONG 0
IBM.PSSP.CSS.ibytes_msw PSSP_LONG 0
IBM.PSSP.CSS.ierrors PSSP_LONG 0
IBM.PSSP.CSS.ipackets_dlt PSSP_LONG 0
IBM.PSSP.CSS.ipackets_drop PSSP_LONG 0
IBM.PSSP.CSS.ipackets_lsw PSSP_LONG 0
IBM.PSSP.CSS.ipackets_msw PSSP_LONG 0
IBM.PSSP.CSS.nobufs PSSP_LONG 0
IBM.PSSP.CSS.obytes_dlt PSSP_LONG 0
IBM.PSSP.CSS.obytes_lsw PSSP_LONG 0
IBM.PSSP.CSS.obytes_msw PSSP_LONG 0
IBM.PSSP.CSS.oerrors PSSP_LONG 0
IBM.PSSP.CSS.opackets_dlt PSSP_LONG 0
IBM.PSSP.CSS.opackets_drop PSSP_LONG 0
IBM.PSSP.CSS.opackets_lsw PSSP_LONG 0
IBM.PSSP.CSS.opackets_msw PSSP_LONG 0
IBM.PSSP.CSS.recvintr_dlt PSSP_LONG 0
IBM.PSSP.CSS.recvintr_lsw PSSP_LONG 0
IBM.PSSP.CSS.recvintr_msw PSSP_LONG 0
IBM.PSSP.CSS.xmitintr_dlt PSSP_LONG 0
IBM.PSSP.CSS.xmitintr_lsw PSSP_LONG 0
IBM.PSSP.CSS.xmitintr_msw PSSP_LONG 0
IBM.PSSP.CSS.xmitque_cur PSSP_LONG 0
IBM.PSSP.CSS.xmitque_max PSSP_LONG 0
IBM.PSSP.CSS.xmitque_ovf PSSP_LONG 0
IBM.PSSP.HARMLD.err_count PSSP_LONG 0
IBM.PSSP.HARMLD.mgrs_conn PSSP_LONG 0
IBM.PSSP.HARMLD.refresh_cntr PSSP_LONG 0
IBM.PSSP.LL.SCHEDD.current_jobs PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.failed_connections PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.failed_in_transactions PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.failed_out_transactions PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.jobs_idle PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.jobs_pending PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.jobs_running PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.jobs_starting PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.total_connections PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.total_in_transactions PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.total_jobs_completed PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.total_jobs_rejected PSSP_LLSCHED 17
```

```
IBM.PSSP.LL.SCHEDD.total_jobs_removed PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.total_jobs_submitted PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.total_jobs_vacated PSSP_LLSCHED 17
IBM.PSSP.LL.SCHEDD.total_out_transactions PSSP_LLSCHED 17
IBM.PSSP.LL.STARTD.current_jobs PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.failed_connections PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.failed_in_transactions PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.failed_out_transactions PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.jobs_pending PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.jobs_running PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.jobs_suspended PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.total_connections PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.total_in_transactions PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.total_jobs_completed PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.total_jobs_received PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.total_jobs_rejected PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.total_jobs_removed PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.total_jobs_suspended PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.total_jobs_vacated PSSP_LLSTART 16
IBM.PSSP.LL.STARTD.total_out_transactions PSSP_LLSTART 16
IBM.PSSP.Membership.LANAdapter.state PSSP_ADAPTER 2
IBM.PSSP.Membership.Node.state PSSP_LONG 0
IBM.PSSP.PRCRS.procs_online PSSP_LONG 0
IBM.PSSP.Prog.pcount PSSP_PROCESS 3
IBM.PSSP.Prog.xpcount PSSP_PROCESS 3
IBM.PSSP.Response.Host.state PSSP_LONG 0
IBM.PSSP.Response.Switch.state PSSP_LONG 0
IBM.PSSP.SP_HW.Frame.amp_ARange PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.amp_BRange PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.amp_CRange PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.amp_DRange PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.applicationLevel PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.codeVersion PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.controllerIDMismatch PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.controllerResponds PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.controllerTail PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.controllerTail1LED PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.controllerTail2LED PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.controllerTailActive PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.daemonPollRate PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.feepromEmpty PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.feepromEraseCount PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frACLED PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frDCLED PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frNoPowerModA PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frNoPowerModB PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frNoPowerModC PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frNoPowerModD PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frNodeComm PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerDC_A PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerDC_B PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerDC_C PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerDC_D PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerLED PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerModAbad PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerModBbad PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerModCbad PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerModDbad PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerMods PSSP_Frame 6
```

```
IBM.PSSP.SP_HW.Frame.frPowerOff PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerOff_A PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerOff_B PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerOff_C PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frPowerOff_D PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frRS232Active PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frRS232Active1 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frRS232Active2 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.frTTNodeComm PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen1 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen10 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen11 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen12 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen13 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen14 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen15 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen16 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen2 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen3 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen4 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen5 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen6 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen7 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen8 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodeLinkOpen9 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail1 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail10 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail11 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail12 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail13 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail14 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail15 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail16 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail17 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail2 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail3 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail4 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail5 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail6 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail7 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail8 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.nodefail9 PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.rs232CTS PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.rs232DCD PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.serialNum PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.supFailure PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.tempRange PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.type PSSP_Frame 6
IBM.PSSP.SP_HW.Frame.voltP48Range PSSP_Frame 6
IBM.PSSP.SP_HW.Node.P2_5dPresent PSSP_LONG 0
IBM.PSSP.SP_HW.Node.P48OK PSSP_LONG 0
IBM.PSSP.SP_HW.Node.P48Off10Sec PSSP_LONG 0
IBM.PSSP.SP_HW.Node.P4dPresent PSSP_LONG 0
IBM.PSSP.SP_HW.Node.P5DCok PSSP_LONG 0
IBM.PSSP.SP_HW.Node.codeVersion PSSP_LONG 0
IBM.PSSP.SP_HW.Node.currentShutdown PSSP_LONG 0
IBM.PSSP.SP_HW.Node.envLED PSSP_LONG 0
IBM.PSSP.SP_HW.Node.fanfail1 PSSP_LONG 0
IBM.PSSP.SP_HW.Node.fanfail1d PSSP_LONG 0
```

```
IBM.PSSP.SP_HW.Node.fanfail2 PSSP_LONG 0
IBM.PSSP.SP_HW.Node.fanfail2d PSSP_LONG 0
IBM.PSSP.SP_HW.Node.fanfail3 PSSP_LONG 0
IBM.PSSP.SP_HW.Node.fanfail3d PSSP_LONG 0
IBM.PSSP.SP_HW.Node.fanfail4 PSSP_LONG 0
IBM.PSSP.SP_HW.Node.fanfail4d PSSP_LONG 0
IBM.PSSP.SP_HW.Node.fanfail5d PSSP_LONG 0
IBM.PSSP.SP_HW.Node.keyModeSwitch PSSP_LONG 0
IBM.PSSP.SP_HW.Node.lcd1 PSSP_STRING 4
IBM.PSSP.SP_HW.Node.lcd1flash PSSP_LONG 0
IBM.PSSP.SP_HW.Node.lcd2 PSSP_STRING 4
IBM.PSSP.SP_HW.Node.lcd2flash PSSP_LONG 0
IBM.PSSP.SP_HW.Node.memoryProtect PSSP_LONG 0
IBM.PSSP.SP_HW.Node.nodePower PSSP_LONG 0
IBM.PSSP.SP_HW.Node.nodePowerOn10Sec PSSP_LONG 0
IBM.PSSP.SP_HW.Node.powerLED PSSP_LONG 0
IBM.PSSP.SP_HW.Node.s1PortDTR PSSP_LONG 0
IBM.PSSP.SP_HW.Node.serialLinkOpen PSSP_LONG 0
IBM.PSSP.SP_HW.Node.serialNum PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownN12High PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownN12Low PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP12High PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP12Low PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP2_5dHigh PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP2_5dLow PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP48Low PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP4High PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP4Low PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP4dHigh PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP4dLow PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP5High PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP5Low PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP5iHigh PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP5iLow PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP5mHigh PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownP5mLow PSSP_LONG 0
IBM.PSSP.SP_HW.Node.shutdownTemp PSSP_LONG 0
IBM.PSSP.SP_HW.Node.smpDiagLEDoff PSSP_LONG 0
IBM.PSSP.SP_HW.Node.smpPowerLEDoff PSSP_LONG 0
IBM.PSSP.SP_HW.Node.tempRange PSSP_LONG 0
IBM.PSSP.SP_HW.Node.type PSSP_LONG 0
IBM.PSSP.SP_HW.Node.voltN12Range PSSP_LONG 0
IBM.PSSP.SP_HW.Node.voltP12Range PSSP_LONG 0
IBM.PSSP.SP_HW.Node.voltP2_5dRange PSSP_LONG 0
IBM.PSSP.SP_HW.Node.voltP48Range PSSP_LONG 0
IBM.PSSP.SP_HW.Node.voltP4Range PSSP_LONG 0
IBM.PSSP.SP_HW.Node.voltP4dRange PSSP_LONG 0
IBM.PSSP.SP_HW.Node.voltP5Range PSSP_LONG 0
IBM.PSSP.SP_HW.Node.voltP5iRange PSSP_LONG 0
IBM.PSSP.SP_HW.Node.voltP5mRange PSSP_LONG 0
IBM.PSSP.SP_HW.Switch.P48OK PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.P48Off10Sec PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.applicationLevel PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.chip PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.chipClkMissing PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.chipReceivedInit PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.chipSelftest PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.clockSource PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.codeVersion PSSP_Switch 7
```

```
IBM.PSSP.SP_HW.Switch.envLED PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.epromErase PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.epromProgram PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.fanfail1 PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.fanfail2 PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.fanfail3 PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.fanfail4 PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.fanfail5 PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.feepromEmpty PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.feepromEraseCount PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.mux PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.nodePower PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.nodePowerOn10Sec PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.osc PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.pll PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.port PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.portClkMissing PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.powerLED PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.powerOnReset PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.powerS1 PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.powerS2 PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.ps1Fail PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.ps1FuseGoodRange PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.ps1PowerGoodRange PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.ps2Fail PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.ps2FuseGoodRange PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.ps2PowerGoodRange PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.psParallelFail PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.psParallelRange PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.recPortNotTuned PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.sendPortNotTuned PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.serialNum PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownN5High PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownN5Low PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownOC PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownP12High PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownP12Low PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownP3_3High PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownP3_3Low PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownP48Low PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownP5High PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownP5Low PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.shutdownTemp PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.subtype PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.supFailure PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.synchReset PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.tempRange PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.type PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.voltN5Range PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.voltP12Range PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.voltP3_3Range PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.voltP48Range PSSP_Switch 7
IBM.PSSP.SP_HW.Switch.voltP5Range PSSP_Switch 7
IBM.PSSP.VSD.bytes_read PSSP_VSD 8
IBM.PSSP.VSD.bytes_write PSSP_VSD 8
IBM.PSSP.VSD.cache_hits PSSP_VSD 8
IBM.PSSP.VSD.client_req_read PSSP_VSD 8
IBM.PSSP.VSD.client_req_write PSSP_VSD 8
IBM.PSSP.VSD.local_req_read PSSP_VSD 8
```

```
IBM.PSSP.VSD.local_req_write PSSP_VSD 8
IBM.PSSP.VSD.physical_req_read PSSP_VSD 8
IBM.PSSP.VSD.physical_req_write PSSP_VSD 8
IBM.PSSP.VSD.remote_req_read PSSP_VSD 8
IBM.PSSP.VSD.remote_req_write PSSP_VSD 8
IBM.PSSP.VSD.server PSSP_VSD 8
IBM.PSSP.VSD.state PSSP_VSD 8
IBM.PSSP.VSDdrv.1_retry_count PSSP_LONG 0
IBM.PSSP.VSDdrv.2_retry_count PSSP_LONG 0
IBM.PSSP.VSDdrv.3_retry_count PSSP_LONG 0
IBM.PSSP.VSDdrv.4_retry_count PSSP_LONG 0
IBM.PSSP.VSDdrv.5_retry_count PSSP_LONG 0
IBM.PSSP.VSDdrv.6_retry_count PSSP_LONG 0
IBM.PSSP.VSDdrv.7_retry_count PSSP_LONG 0
IBM.PSSP.VSDdrv.8_retry_count PSSP_LONG 0
IBM.PSSP.VSDdrv.9_retry_count PSSP_LONG 0
IBM.PSSP.VSDdrv.avg_buddy_wait PSSP_REAL 1
IBM.PSSP.VSDdrv.buddy_buffer_shortage PSSP_LONG 0
IBM.PSSP.VSDdrv.cache_shortage PSSP_LONG 0
IBM.PSSP.VSDdrv.comm_buf_shortage PSSP_LONG 0
IBM.PSSP.VSDdrv.indirect_io PSSP_LONG 0
IBM.PSSP.VSDdrv.pbuf_shortage PSSP_LONG 0
IBM.PSSP.VSDdrv.rejected_no_buddy_buffer PSSP_LONG 0
IBM.PSSP.VSDdrv.rejected_requests PSSP_LONG 0
IBM.PSSP.VSDdrv.rejected_responds PSSP_LONG 0
IBM.PSSP.VSDdrv.request_block_shortage PSSP_LONG 0
IBM.PSSP.VSDdrv.request_rework PSSP_LONG 0
IBM.PSSP.VSDdrv.timeout_error PSSP_LONG 0
IBM.PSSP.aixos.CPU.glidle PSSP_REAL 1
IBM.PSSP.aixos.CPU.glkern PSSP_REAL 1
IBM.PSSP.aixos.CPU.gluser PSSP_REAL 1
IBM.PSSP.aixos.CPU.glwait PSSP_REAL 1
IBM.PSSP.aixos.Disk.busy PSSP_Disk 9
IBM.PSSP.aixos.Disk.rblk PSSP_Disk 9
IBM.PSSP.aixos.Disk.wblk PSSP_Disk 9
IBM.PSSP.aixos.Disk.xfer PSSP_Disk 9
IBM.PSSP.aixos.FS.%nodesused PSSP_FS 10
IBM.PSSP.aixos.FS.%totused PSSP_FS 10
IBM.PSSP.aixos.LAN.rcverrors PSSP_Network 5
IBM.PSSP.aixos.LAN.recvdrops PSSP_Network 5
IBM.PSSP.aixos.LAN.xmitdrops PSSP_Network 5
IBM.PSSP.aixos.LAN.xmiterrors PSSP_Network 5
IBM.PSSP.aixos.LAN.xmitovfl PSSP_Network 5
IBM.PSSP.aixos.Mem.Kmem.calls PSSP_Kmem 11
IBM.PSSP.aixos.Mem.Kmem.failures PSSP_Kmem 11
IBM.PSSP.aixos.Mem.Kmem.inuse PSSP_Kmem 11
IBM.PSSP.aixos.Mem.Kmem.memuse PSSP_Kmem 11
IBM.PSSP.aixos.Mem.Real.%free PSSP_LONG 0
IBM.PSSP.aixos.Mem.Real.%pinned PSSP_LONG 0
IBM.PSSP.aixos.Mem.Real.numfrb PSSP_LONG 0
IBM.PSSP.aixos.Mem.Real.size PSSP_LONG 0
IBM.PSSP.aixos.Mem.Virt.pagein PSSP_LONG 0
IBM.PSSP.aixos.Mem.Virt.pageout PSSP_LONG 0
IBM.PSSP.aixos.Mem.Virt.pagexct PSSP_LONG 0
IBM.PSSP.aixos.Mem.Virt.pgspgin PSSP_LONG 0
IBM.PSSP.aixos.Mem.Virt.pgspgout PSSP_LONG 0
IBM.PSSP.aixos.PagSp.%totalfree PSSP_REAL 1
IBM.PSSP.aixos.PagSp.%totalused PSSP_REAL 1
IBM.PSSP.aixos.PagSp.totalfree PSSP_LONG 0
```

```
IBM.PSSP.aixos.PagSp.totalsize PSSP_LONG 0
IBM.PSSP.aixos.Proc.runque PSSP_REAL 1
IBM.PSSP.aixos.Proc.swpque PSSP_REAL 1
IBM.PSSP.aixos.VG.free PSSP_VG 12
IBM.PSSP.aixos.cpu.idle PSSP_REAL 1
IBM.PSSP.aixos.cpu.kern PSSP_REAL 1
IBM.PSSP.aixos.cpu.user PSSP_REAL 1
IBM.PSSP.aixos.cpu.wait PSSP_REAL 1
IBM.PSSP.aixos.pagsp.%free PSSP_PageSpace 14
IBM.PSSP.aixos.pagsp.size PSSP_PageSpace 14
IBM.PSSP.pm.Errlog PSSP_Errlog 15
IBM.PSSP.pm.User_state1 PSSP_STRING 4
IBM.PSSP.pm.User_state10 PSSP_STRING 4
IBM.PSSP.pm.User_state11 PSSP_STRING 4
IBM.PSSP.pm.User_state12 PSSP_STRING 4
IBM.PSSP.pm.User_state13 PSSP_STRING 4
IBM.PSSP.pm.User_state14 PSSP_STRING 4
IBM.PSSP.pm.User_state15 PSSP_STRING 4
IBM.PSSP.pm.User_state16 PSSP_STRING 4
IBM.PSSP.pm.User_state2 PSSP_STRING 4
IBM.PSSP.pm.User_state3 PSSP_STRING 4
IBM.PSSP.pm.User_state4 PSSP_STRING 4
IBM.PSSP.pm.User_state5 PSSP_STRING 4
IBM.PSSP.pm.User_state6 PSSP_STRING 4
IBM.PSSP.pm.User_state7 PSSP_STRING 4
IBM.PSSP.pm.User_state8 PSSP_STRING 4
IBM.PSSP.pm.User_state9 PSSP_STRING 4
```

Appendix E. Contents of SNMP Adapter Class Definition Statements

This appendix contains a listing of the Class Definition Statement file used in the example described in 4.3, “Using the TME 10 T/EC SNMP Adapter” on page 45.

```
#########################
## SP Traps
##
## The following classes are defined to recognize SNMP traps
##  with the trapid (ibmSPEMEventID or SNMP specific trap)
##  set as follows:
##        trapid=1   SP_SNMP_FATAL_TRAP
##        trapid=2   SP_SNMP_CRITICAL_TRAP
##        trapid=3   SP_SNMP_MINOR_TRAP
##        trapid=4   SP_SNMP_WARNING_TRAP
##        trapid=5   SP_SNMP_HARMLESS_TRAP
##        trapid=??  SP_SNMP_UNKNOWN_TRAP
## All ibmSPEM* values in the MIB are set and passed to TEC
## The corresponding baroc file contains the same classes and
## variables.
##
## To add your own specific definitions, copy one of the existing
## classes and place it BEFORE the SP_SNMP_UNKNOWN_TRAP class.
## Change the $SPECIFIC selection test to contain the value of
## your unique trapid as set in the PSSP event.  Make the required
## changes to the baroc file that is loaded into TEC.
#########################
CLASS SP_SNMP_FATAL_TRAP
  SELECT
    1: ATTR(=,$ENTERPRISE), VALUE(PREFIX, "1.3.6.1.4.1.2.6.117") ;
    2: $SPECIFIC = 1;
    3: ATTR(=, "ibmSPEMEventID" ) ;
    4: ATTR(=, "ibmSPEMEventFlags" ) ;
    5: ATTR(=, "ibmSPEMEventTime" ) ;
    6: ATTR(=, "ibmSPEMEventLocation" ) ;
    7: ATTR(=, "ibmSPEMEventPartitionAddress" ) ;
    8: ATTR(=, "ibmSPEMEventVarsTableName" ) ;
    9: ATTR(=, "ibmSPEMEventVarsTableInstanceID" ) ;
   10: ATTR(=, "ibmSPEMEventVarName" ) ;
   11: ATTR(=, "ibmSPEMEventVarValueInstanceVector" ) ;
   12: ATTR(=, "ibmSPEMEventVarValuesTableInstanceID" ) ;
   13: ATTR(=, "ibmSPEMEventVarValue" ) ;
   14: ATTR(=, "ibmSPEMEventPredicate" ) ;
  FETCH
    1: IPNAME($SOURCE_ADDR) ;
  MAP
    hostname = $F1 ;
    enterprise = $ENTERPRISE ;
    pssp_EMEventID = $V3 ;
    pssp_EMEventFlags = $V4 ;
    pssp_EMEventTime = $V5 ;
    pssp_EMEventLocation = $V6 ;
    pssp_EMEventPartitionAddress = $V7 ;
    pssp_EMEventVarsTableName = $V8 ;
    pssp_EMEventVarsTableInstanceID = $V9 ;
    pssp_EMEventVarName = $V10 ;
    pssp_EMEventVarValueInstanceVector = $V11 ;
    pssp_EMEventVarValuesTableInstanceID = $V12 ;
```

```
    pssp_EMEventVarValue = $V13 ;
    pssp_EMEventPredicate = $V14 ;
END

CLASS SP_SNMP_CRITICAL_TRAP
  SELECT
    1: ATTR(=,$ENTERPRISE), VALUE(PREFIX, "1.3.6.1.4.1.2.6.117") ;
    2: $SPECIFIC = 2;
    3: ATTR(=, "ibmSPEMEventID" ) ;
    4: ATTR(=, "ibmSPEMEventFlags" ) ;
    5: ATTR(=, "ibmSPEMEventTime" ) ;
    6: ATTR(=, "ibmSPEMEventLocation" ) ;
    7: ATTR(=, "ibmSPEMEventPartitionAddress" ) ;
    8: ATTR(=, "ibmSPEMEventVarsTableName" ) ;
    9: ATTR(=, "ibmSPEMEventVarsTableInstanceID" ) ;
   10: ATTR(=, "ibmSPEMEventVarName" ) ;
   11: ATTR(=, "ibmSPEMEventVarValueInstanceVector" ) ;
   12: ATTR(=, "ibmSPEMEventVarValuesTableInstanceID" ) ;
   13: ATTR(=, "ibmSPEMEventVarValue" ) ;
   14: ATTR(=, "ibmSPEMEventPredicate" ) ;
  FETCH
    1: IPNAME($SOURCE_ADDR) ;
  MAP
    hostname = $F1 ;
    enterprise = $ENTERPRISE ;
    pssp_EMEventID = $V3 ;
    pssp_EMEventFlags = $V4 ;
    pssp_EMEventTime = $V5 ;
    pssp_EMEventLocation = $V6 ;
    pssp_EMEventPartitionAddress = $V7 ;
    pssp_EMEventVarsTableName = $V8 ;
    pssp_EMEventVarsTableInstanceID = $V9 ;
    pssp_EMEventVarName = $V10 ;
    pssp_EMEventVarValueInstanceVector = $V11 ;
    pssp_EMEventVarValuesTableInstanceID = $V12 ;
    pssp_EMEventVarValue = $V13 ;
    pssp_EMEventPredicate = $V14 ;
END

CLASS SP_SNMP_MINOR_TRAP
  SELECT
    1: ATTR(=,$ENTERPRISE), VALUE(PREFIX, "1.3.6.1.4.1.2.6.117") ;
    2: $SPECIFIC = 3;
    3: ATTR(=, "ibmSPEMEventID" ) ;
    4: ATTR(=, "ibmSPEMEventFlags" ) ;
    5: ATTR(=, "ibmSPEMEventTime" ) ;
    6: ATTR(=, "ibmSPEMEventLocation" ) ;
    7: ATTR(=, "ibmSPEMEventPartitionAddress" ) ;
    8: ATTR(=, "ibmSPEMEventVarsTableName" ) ;
    9: ATTR(=, "ibmSPEMEventVarsTableInstanceID" ) ;
   10: ATTR(=, "ibmSPEMEventVarName" ) ;
   11: ATTR(=, "ibmSPEMEventVarValueInstanceVector" ) ;
   12: ATTR(=, "ibmSPEMEventVarValuesTableInstanceID" ) ;
   13: ATTR(=, "ibmSPEMEventVarValue" ) ;
   14: ATTR(=, "ibmSPEMEventPredicate" ) ;
  FETCH
    1: IPNAME($SOURCE_ADDR) ;
  MAP
    hostname = $F1 ;
```

```
    enterprise = $ENTERPRISE ;
    pssp_EMEventID = $V3 ;
    pssp_EMEventFlags = $V4 ;
    pssp_EMEventTime = $V5 ;
    pssp_EMEventLocation = $V6 ;
    pssp_EMEventPartitionAddress = $V7 ;
    pssp_EMEventVarsTableName = $V8 ;
    pssp_EMEventVarsTableInstanceID = $V9 ;
    pssp_EMEventVarName = $V10 ;
    pssp_EMEventVarValueInstanceVector = $V11 ;
    pssp_EMEventVarValuesTableInstanceID = $V12 ;
    pssp_EMEventVarValue = $V13 ;
    pssp_EMEventPredicate = $V14 ;
END

CLASS SP_SNMP_WARNING_TRAP
  SELECT
    1: ATTR(=,$ENTERPRISE), VALUE(PREFIX, "1.3.6.1.4.1.2.6.117") ;
    2: $SPECIFIC = 4;
    3: ATTR(=, "ibmSPEMEventID" ) ;
    4: ATTR(=, "ibmSPEMEventFlags" ) ;
    5: ATTR(=, "ibmSPEMEventTime" ) ;
    6: ATTR(=, "ibmSPEMEventLocation" ) ;
    7: ATTR(=, "ibmSPEMEventPartitionAddress" ) ;
    8: ATTR(=, "ibmSPEMEventVarsTableName" ) ;
    9: ATTR(=, "ibmSPEMEventVarsTableInstanceID" ) ;
   10: ATTR(=, "ibmSPEMEventVarName" ) ;
   11: ATTR(=, "ibmSPEMEventVarValueInstanceVector" ) ;
   12: ATTR(=, "ibmSPEMEventVarValuesTableInstanceID" ) ;
   13: ATTR(=, "ibmSPEMEventVarValue" ) ;
   14: ATTR(=, "ibmSPEMEventPredicate" ) ;
  FETCH
    1: IPNAME($SOURCE_ADDR) ;
  MAP
    hostname = $F1 ;
    enterprise = $ENTERPRISE ;
    pssp_EMEventID = $V3 ;
    pssp_EMEventFlags = $V4 ;
    pssp_EMEventTime = $V5 ;
    pssp_EMEventLocation = $V6 ;
    pssp_EMEventPartitionAddress = $V7 ;
    pssp_EMEventVarsTableName = $V8 ;
    pssp_EMEventVarsTableInstanceID = $V9 ;
    pssp_EMEventVarName = $V10 ;
    pssp_EMEventVarValueInstanceVector = $V11 ;
    pssp_EMEventVarValuesTableInstanceID = $V12 ;
    pssp_EMEventVarValue = $V13 ;
    pssp_EMEventPredicate = $V14 ;
END

CLASS SP_SNMP_HARMLESS_TRAP
  SELECT
    1: ATTR(=,$ENTERPRISE), VALUE(PREFIX, "1.3.6.1.4.1.2.6.117") ;
    2: $SPECIFIC = 5;
    3: ATTR(=, "ibmSPEMEventID" ) ;
    4: ATTR(=, "ibmSPEMEventFlags" ) ;
    5: ATTR(=, "ibmSPEMEventTime" ) ;
    6: ATTR(=, "ibmSPEMEventLocation" ) ;
    7: ATTR(=, "ibmSPEMEventPartitionAddress" ) ;
```

```
    8: ATTR(=, "ibmSPEMEventVarsTableName" ) ;
    9: ATTR(=, "ibmSPEMEventVarsTableInstanceID" ) ;
   10: ATTR(=, "ibmSPEMEventVarName" ) ;
   11: ATTR(=, "ibmSPEMEventVarValueInstanceVector" ) ;
   12: ATTR(=, "ibmSPEMEventVarValuesTableInstanceID" ) ;
   13: ATTR(=, "ibmSPEMEventVarValue" ) ;
   14: ATTR(=, "ibmSPEMEventPredicate" ) ;
  FETCH
    1: IPNAME($SOURCE_ADDR) ;
  MAP
    hostname = $F1 ;
    enterprise = $ENTERPRISE ;
    pssp_EMEventID = $V3 ;
    pssp_EMEventFlags = $V4 ;
    pssp_EMEventTime = $V5 ;
    pssp_EMEventLocation = $V6 ;
    pssp_EMEventPartitionAddress = $V7 ;
    pssp_EMEventVarsTableName = $V8 ;
    pssp_EMEventVarsTableInstanceID = $V9 ;
    pssp_EMEventVarName = $V10 ;
    pssp_EMEventVarValueInstanceVector = $V11 ;
    pssp_EMEventVarValuesTableInstanceID = $V12 ;
    pssp_EMEventVarValue = $V13 ;
    pssp_EMEventPredicate = $V14 ;
END

CLASS SP_SNMP_UNKNOWN_TRAP
  SELECT
    1: ATTR(=,$ENTERPRISE), VALUE(PREFIX, "1.3.6.1.4.1.2.6.117") ;
    2: ATTR(=, "ibmSPEMEventID" ) ;
    3: ATTR(=, "ibmSPEMEventFlags" ) ;
    4: ATTR(=, "ibmSPEMEventTime" ) ;
    5: ATTR(=, "ibmSPEMEventLocation" ) ;
    6: ATTR(=, "ibmSPEMEventPartitionAddress" ) ;
    7: ATTR(=, "ibmSPEMEventVarsTableName" ) ;
    8: ATTR(=, "ibmSPEMEventVarsTableInstanceID" ) ;
    9: ATTR(=, "ibmSPEMEventVarName" ) ;
   10: ATTR(=, "ibmSPEMEventVarValueInstanceVector" ) ;
   11: ATTR(=, "ibmSPEMEventVarValuesTableInstanceID" ) ;
   12: ATTR(=, "ibmSPEMEventVarValue" ) ;
   13: ATTR(=, "ibmSPEMEventPredicate" ) ;
  FETCH
    1: IPNAME($SOURCE_ADDR) ;
  MAP
    hostname = $F1 ;
    enterprise = $ENTERPRISE ;
    pssp_EMEventID = $V2 ;
    pssp_EMEventFlags = $V3 ;
    pssp_EMEventTime = $V4 ;
    pssp_EMEventLocation = $V5 ;
    pssp_EMEventPartitionAddress = $V6 ;
    pssp_EMEventVarsTableName = $V7 ;
    pssp_EMEventVarsTableInstanceID = $V8 ;
    pssp_EMEventVarName = $V9  ;
    pssp_EMEventVarValueInstanceVector = $V10 ;
    pssp_EMEventVarValuesTableInstanceID = $V11 ;
    pssp_EMEventVarValue = $V12 ;
    pssp_EMEventPredicate = $V13 ;
END
```

Appendix F. Logfile CDS

This appendix contains an example of a class definition statement as used in the example in 4.4, “Using the TME 10 T/EC Logfile Adapter” on page 58.

```
FORMAT EM_Prog FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.Prog.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END

FORMAT EM_PRCRS FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.PRCRS.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END

FORMAT EM_CSS FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.CSS.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END

FORMAT EM_HARMLD FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.HARMLD.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
```

```
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END

FORMAT EM_LL FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.LL.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END

FORMAT EM_Membership FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.Membership.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END

FORMAT EM_Response FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.Response.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END
```

```
FORMAT EM_SP_HW FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.SP_HW.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END

FORMAT EM_VSD FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.VSD.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END

FORMAT EM_aixos FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.aixos.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END

FORMAT EM_pm FOLLOWS Log_EM
%t %s %s: PMAN_HANDLE=%s PMAN_IVECTOR=%s PMAN_LOCATION=%s PMAN_PRED=%s
PMAN_PRINCIPAL=%s PMAN_RVCOUNT=%s PMAN_RVNAME=IBM.PSSP.pm.%s
PMAN_RVTYPE=%s PMAN_TIME=%s+ PMAN_RVFIELDO=%s+
PMAN_HANDLE $4
PMAN_IVECTOR $5
PMAN_LOCATION $6
PMAN_PRED $7
PMAN_PRINCIPAL $8
PMAN_RVCOUNT $9
PMAN_RVNAME $10
```

```
PMAN_RVTYPE $11
PMAN_TIME $12
PMAN_RVFIELDO $13
END
```

Appendix G. Sample Task Library Listings

This appendix contains sample task library listings for two task libraries. The first contains general RS/6000 SP tasks, and the second contains tasks that are specific to maintaining the switch.

See 5.3, “Using the Task Library Language” on page 97 for instructions on importing these task libraries into your TME 10 Desktop.

G.1 SP Task Library Source Listing

The following source listing contains an example of a task library that performs some common SP tasks. It includes operations such as:

SPData	List data for the SP using the splstdata command.
NodeData	List data for the SP node the task is run on using the splstdata command.
RunCommand	Run any specified command.
Perspectives	Launch the Perspectives application as a separate process.
LaunchApplication	Launch a specified application as a separate process.
Startup	Run the cstartup command on the control workstation using specified parameters.
Shutdown	Run the cshutdown command on the control workstation using specified parameters.

G.1.1 SPTasks.tll

```
TaskLibrary "SPTasks" {
        Context = ("_!_","*",1);
        Distribute = ("_!_","ALI",1);
        HelpMessage = ("_!_","Conventional Task Library",1);
        Requires = ("_!_",">2.5",1);
        Version = ("_!_","1.0",1);
        ArgLayout inputText {
                Text;
        };

        Task Perspectives {
                Description = ("_!_","Invoke Perspectives as a separate process",1);
                HelpMessage = ("_!_","No Help Available",1);
                Uid = ("_!_","*",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Implementation ("default")
.#!/bin/ksh
.persp="/usr/lpp/ssp/bin/perspectives"
.if [[ ! -a $persp ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH
. . /etc/Tivoli/setup_env.sh
.export DISPLAY=$WD_DISPLAY
```

```
.# We close stdin, stdout and stderr
.exec 0<&-
.exec 1<&-
.exec 2<&-
.$persp &
                ;
        };

        Task NodeData {
                Description = ("_!_","List node data using splstdata command",1);
                HelpMessage = ("_!_","No Help Available",1);
                Uid = ("_!_","*",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Implementation ("default")
.#!/bin/ksh
.# Get the SDR attributes for a node
.node_number="/usr/lpp/ssp/install/bin/node_number"
.lstdata="/usr/lpp/ssp/bin/splstdata"
.if [[ ! -a $node_number ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.this_node=$node_number
.if [[ $this_node = 0 ]] then
.  echo "ERROR:  This is a CWS.  This task can only be run on a node."
.  exit 1
.fi
.$lstdata -G -n -l $this_node
.echo "\n\n"
.$lstdata -G -A -l $this_node
.echo "\n\n"
.$lstdata -G -b -l $this_node
.echo "\n\n"
.$lstdata -G -a -l $this_node
.echo "\n\n"
.$lstdata -G -u -l $this_node
.echo "\n\n"
.$lstdata -G -h -l $this_node
.echo "\n\n"
.$lstdata -G -i -l $this_node
.echo "\n\n"
.$lstdata -G -d -l $this_node
                ;
        };

        Task SPData {
                Description = ("_!_","List SP data using splstdata command",1);
                HelpMessage = ("_!_","No Help Available",1);
                Uid = ("_!_","*",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Implementation ("default")
.#!/bin/ksh
.# Get the SP data for this machine
.lstdata="/usr/lpp/ssp/bin/splstdata"
.if [[ ! -a $lstdata ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
```

```
.fi
.$lstdata -e
.echo "\n\n"
.$lstdata -f
.echo "\n\n"
.$lstdata -p
.echo "\n\n"
.$lstdata -G -n
.echo "\n\n"
.$lstdata -G -s
                ;
        };

        Task RunCommand {
                Description = ("_!_","Execute any command",1);
                HelpMessage = ("_!_","Input the command and any parameters",1);
                Uid = ("_!_","*",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Argument ("_!_","Command to run: ",1) {
                        Layout = ("","inputText",1);
                };
                Implementation ("default")
.#!/bin/ksh
.export PATH=/usr/lpp/ssp/bin:/tivoli/bin/aix4-r1/CUSTOM:$PATH
. . /etc/Tivoli/setup_env.sh
.export DISPLAY=$WD_DISPLAY
.$1
                ;
        };

        Task LaunchApplication {
                Description = ("_!_","Launch an application as a separate process",1);
                HelpMessage = ("_!_","Input the application and parameters.",1);
                Uid = ("_!_","*",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Argument ("_!_","Application to launch: ",1) {
                        Layout = ("","inputText",1);
                };
                Implementation ("default")
.#!/bin/ksh
.export PATH=/usr/lpp/ssp/bin:/tivoli/bin/aix4-r1/CUSTOM:$PATH
. . /etc/Tivoli/setup_env.sh
.export DISPLAY=$WD_DISPLAY
.# We close stdin, stdout and stderr
.exec 0<&-
.exec 1<&-
.exec 2<&-
.$1 &
                ;
        };

        Task Startup {
                Description = ("_!_","Run cstartup command",1);
                HelpMessage = ("_!_","Input parameters to cstartup command",1);
                Uid = ("_!_","root",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
```

```
                Argument ("_!_","Parameters for cstartup command (e.g. node list): ",1) {
                        Layout = ("","inputText",1);
                };
                Implementation ("default")
.#!/bin/ksh
.startup="/usr/lpp/ssp/bin/cstartup"
.if [[ ! -a $startup ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.this_node=/usr/lpp/ssp/install/bin/node_number
.if [[ ! $this_node = 0 ]] then
.  echo "ERROR:  This is a not CWS.  This task can only be run on the CWS."
.  exit 1
.fi
.#  Run cstartup with the specified input
.$startup $1
                ;
        };

        Task Shutdown {
                Description = ("_!_","Run cshutdown command",1);
                HelpMessage = ("_!_","Input parameters to cshutdown command",1);
                Uid = ("_!_","root",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Argument ("_!_","Parameters for cshutdown command (e.g. node list): ",1) {
                        Layout = ("","inputText",1);
                };
                Implementation ("default")
.#!/bin/ksh
.shutdown="/usr/lpp/ssp/bin/cshutdown"
.if [[ ! -a $shutdown ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.this_node=/usr/lpp/ssp/install/bin/node_number
.if [[ ! $this_node = 0 ]] then
.  echo "ERROR:  This is a not CWS.  This task can only be run on the CWS."
.  exit 1
.fi
.#  Run cshutdown with the specified input
.$shutdown $1
                ;
        };
}
```

G.2 Switch Task Library Source Listing

The following source listing contains an example of a task library that performs some common SP switch tasks. It includes operations such as:

SwitchData List data for the SP switch using the splstdata command.

SwitchResponds List the switch responds info using the SDRGetObjects command.

ListPrimaryNode List the primary node information returned by the Eprimary command.

SetPrimaryNode Set the specified oncoming primary and backup nodes using the Eprimary command.

QuiesceSwitch Quiesce the switch for the specified partition.

StartSwitch Start the switch for the specified partition.

FenceNode Fence the specified nodes.

UnfenceNode Unfence the specified nodes.

G.2.1 SwitchTasks.tll

```
TaskLibrary "SwitchTasks" {
        Context = ("_!_","*",1);
        Distribute = ("_!_","ALI",1);
        HelpMessage = ("_!_","Conventional Task Library",1);
        Requires = ("_!_",">2.5",1);
        Version = ("_!_","1.0",1);
        ArgLayout inputText {
                Text;
        };

        ArgLayout Partitions {
                TextChoice Program {
                   Implementation ( "default" )
.#!/bin/ksh
.for i in /usr/lpp/ssp/bin/splst_syspars -n
.do
.  echo $i
.done
                   ;
                };
        };

        Task SwitchResponds {
                Description = ("_!_","List switch responds using SDRGetObjects",1);
                HelpMessage = ("_!_","No Help Available",1);
                Uid = ("_!_","*",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Implementation ("default")
.#!/bin/ksh
.# get the switch responds info from the SDR
.getobjs="/usr/lpp/ssp/bin/SDRGetObjects"
.if [[ ! -a $getobjs ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.$getobjs switch_responds
                ;
        };

        Task SwitchData {
                Description = ("_!_","List switch info using splstdata",1);
                HelpMessage = ("_!_","No Help Available",1);
                Uid = ("_!_","*",1);
                Comments = ("_!_"," ",1);
```

```
                Roles = ("_!_","admin",1);
                Implementation ("default")
.#!/bin/ksh
.# Get the SP data for this machine
.lstdata="/usr/lpp/ssp/bin/splstdata"
.if [[ ! -a $lstdata ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.$lstdata -G -s
                ;
        };

        Task ListPrimaryNode {
                Description = ("_!_","List the switch primary node",1);
                HelpMessage = ("_!_","No Help Available",1);
                Uid = ("_!_","root",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Implementation ("default")
.#!/bin/ksh
.# get the Primary switch node number
.eprimary="/usr/lpp/ssp/bin/Eprimary"
.if [[ ! -a $eprimary ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.#  if this is a TB3 switch (SP switch), will return backup also
.echo "Switch Primary Node:"
.$eprimary
                ;
        };

        Task SetPrimaryNode {
                Description = ("_!_","Set the switch primary node",1);
                HelpMessage = ("_!_","Input primary node and optional backup node",1);
                Uid = ("_!_","root",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Argument ("_!_","New Primary Node: ",1) {
                        Layout = ("","inputText",1);
                };
                Argument ("_!_","New Backup Node: ",1) {
                        Layout = ("","inputText",1);
                };
                Implementation ("default")
.#!/bin/ksh
.eprimary="/usr/lpp/ssp/bin/Eprimary"
.if [[ ! -a $eprimary ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.# set the Primary switch node number
.#  if this is a TB3 switch (SP switch), can set backup also
.if [[ ! -z $2 ]]
.then
.   backup="-backup $2"
.else
.   backup=""
```

```
.fi
.$eprimary $1 $backup
                ;
        };

        Task FenceNode {
                Description = ("_!_","Fence the specified nodes",1);
                HelpMessage = ("_!_","Input node list to be fenced.",1);
                Uid = ("_!_","root",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Argument ("_!_","Fence node list: ",1) {
                        Layout = ("","inputText",1);
                };
                Implementation ("default")
.#!/bin/ksh
.efence="/usr/lpp/ssp/bin/Efence"
.if [[ ! -a $efence ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.#  Fence the specified node regardless of partition
.$efence -G $1
                ;
        };

        Task UnfenceNode {
                Description = ("_!_","Unfence the specified nodes",1);
                HelpMessage = ("_!_","Input node list to be unfenced.",1);
                Uid = ("_!_","root",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Argument ("_!_","Unfence node list: ",1) {
                        Layout = ("","inputText",1);
                };
                Implementation ("default")
.#!/bin/ksh
.eunfence="/usr/lpp/ssp/bin/Eunfence"
.if [[ ! -a $eunfence ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.#  Unfence the specified node regardless of partition
.$eunfence -G $1
                ;
        };

        Task QuiesceSwitch {
                Description = ("_!_","Quiesce the switch in the specified partition",1);
                HelpMessage = ("_!_","Select partition to quiesce switch in",1);
                Uid = ("_!_","root",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Argument ("_!_","Partition to quiesce switch in: ",1) {
                        Layout = ("","Partitions",1);
                };
                Implementation ("default")
.#!/bin/ksh
.equiesce="/usr/lpp/ssp/bin/Equiesce"
```

```
.if [[ ! -a $equiesce ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.# set the current partition
.syspar=$1
.$equiesce
                ;
        };

        Task StartSwitch {
                Description = ("_!_","Start the switch in the specified partition",1);
                HelpMessage = ("_!_","Select partition to start switch in",1);
                Uid = ("_!_","root",1);
                Comments = ("_!_"," ",1);
                Roles = ("_!_","admin",1);
                Argument ("_!_","Partition to start switch in: ",1) {
                        Layout = ("","Partitions",1);
                };
                Implementation ("default")
.#!/bin/ksh
.estart="/usr/lpp/ssp/bin/Estart"
.if [[ ! -a $estart ]]; then
.  echo "ERROR:  This is a non-SP machine.  This task can only be run on an SP."
.  exit 1
.fi
.# set the current partition
.syspar=$1
.$estart
                ;
        };


}
```

Appendix H. Contents of AEF Customization Scripts

This appendix contains all AEF customization source files.

H.1 Dialog Listings

The dialog source files are listed in the following sections. Note that dialogs specific to the SP nodes are prefixed with sp_node, and the dialogs specific to the control workstation are prefixed by sp_cws.

- sp.run_dialog. This dialog is used both by the control workstation and sp node objects when the user clicks on the **run** button on the properties window.
- sp_cws.cw_attributes_dialog. This dialog displays the SDR attributes of the control workstation.
- sp_cws.modify_attribute_dialog. This dialog appears when the user double clicks in one of the attributes displayed in sp_cws.cw_attributes_dialog in order to modify them. This operation is only allowed to an administrator with the role of super.
- sp_cws.node_response. This dialog lists all the SP nodes controlled by the control workstation objects, indicating whether they are managed by Tivoli (wping) and whether they are responding.
- sp_cws.parent_dialog. This is the customized parent dialog.
- sp_cws.parent_dialog.custom. This dialog is similar to the customized parent dialog, but it indicates customizations were made. The user should utilize this version when updating a parent_dialog with her own customizations.
- sp_cws.run_dialog_nodes. This dialog is similar to sp.run_dialog, but it takes as target nodes a list of selected nodes from the parent_dialog of the control workstation.
- sp_cws.sp_applications. This dialog displays a set of buttons to launch SP-specific applications, SP Perspectives and the SP System Monitor GUI.
- sp_node.modify_attribute_dialog. This dialog is similar to sp_cws.modify_attribute_dialog, but is used for the SP node object.
- sp_node.node_attributes_dialog. This dialog is similar to sp_cws.cw_attributes_dialog, but it is specific to the SP node object.
- sp_node.parent_dialog. This dialog is a customized version of the parent dialog of the managed node and is used to accommodate the specifics of an SP node.
- sp_node.parent_dialog.custom. This dialog is similar to sp_node.parent.dialog, but it indicates where the customizations were made. The user should utilize this version when updating a parent dialog with her own customizations.

H.1.1 sp.run_command.ksh

```
#!/bin/ksh
##############################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp.run_command.ksh
# CALLED FROM DIALOG(S): sp.run_dialog
# INPUT: Command (with arguments) to be executed.
# COMMENTS: Executes command (with arguments) and displays resulting
#           standard output, standard error, and exit status in a
#           'wdispmsg' dialog.
#
##############################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

output=/usr/lpp/ssp/install/bin/spcmdrc $* 2>&1

wdispmsg "$output"

exit $?
```

H.1.2 sp.run_command_driver.ksh

```
#!/bin/ksh
##############################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp.run_command_driver.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog, sp_node.parent_dialog
# INPUT: none
# COMMENTS: Posts dialog which accepts command (with arguments) to be
#           executed.
#
##############################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

wpostdialog sp.run_dialog

exit $?
```

H.1.3 sp_cws.check_node_response.ksh

```
#!/bin/ksh
#######################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.check_node_response.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog
# INPUT: whitespace-separated list of one or more node numbers
# COMMENTS: Posts 'sp_cws.node_response' dialog which is a list
#           containing (for each row) node number, whether or not
#           the node can be ping'd (i.e. responds), and whether or
#           not the node is considered to be managed by Tivoli
#           (i.e. has a running 'oserv' daemon).
#
#######################################################################

function doit
{
        typeset -i rc=0

        nodeNumber=$1

        hostname=/usr/lpp/ssp/bin/SDRGetObjects -x -G Node node_number==$nodeNumber
                  reliable_hostname

        if [ -z "$hostname" ]
        then
           echo "[ERROR: hostname not found for node '$nodeNumber']  []\c"
           return 0
        fi

        pingResult=/etc/ping -q -c 1 $hostname 2>/dev/null | grep "received,"

        if [ -z "$pingResult" ]
        then
           pingResult="[NOT FOUND]"
           rc=1
        else
           set $pingResult
              if [ "$4" = "0" ]
              then
                 pingResult="[NOT RESPONDING]"
                 rc=2
              else
                 pingResult="[RESPONDS]"
                 fi
        fi

#        wpingResult=wping $hostname 5 2>&1
#
#        # there are actually three possibilities, here, but we're doing
#        # only two... the three are:
#        # 1) managed and object dispatcher is alive
#        # 2) managed and no response from object dispatcher
#        # 3) no such instance of ManagedNode found
#
```

```
#        wresult=echo "$wpingResult" | grep " alive"
#
#        if [ -n "$wresult" ]
#                then    wpingResult="[MANAGED]"
#                else        wpingResult="[NOT MANAGED]"
#        fi

# Do the following instead of 'wping' (see code commented out, above)
# because 'wping' only seems to work if executed on the TME server machine

        if [ "$pingResult" = "[RESPONDS]" ]
        then
           wresult=/usr/lpp/ssp/bin/dsh -w $hostname ps -u root \| grep oserv
              if [ -n "$wresult" ]
              then
                 wpingResult="[MANAGED]"
              else
                 wpingResult="[NOT MANAGED]"
              fi
        else
           wpingResult="[UNKNOWN]"
        fi

        awk -v nodeNumber=$nodeNumber -v hostname=$hostname \
            -v pingResult="$pingResult" -v wpingResult="$wpingResult" '

        BEGIN {
        printf("\"%5s   %-28s%-19s%s\"{\"\"}", nodeNumber, hostname, pingResult,
               wpingResult);
        }

        {
        }' </dev/null

        return $rc
}

. /etc/Tivoli/setup_env.sh

if [ -z "$*" -o "$*" = "__unchanged__" ]
then
   wdispmsg "You must first select one or more nodes..."
   exit 0
fi

typeset -i num=$#

while [[ $num != 0 ]]
do
        result=$resultdoit $1
        if [[ $num != 1 ]]
        then        result="$result,"
        fi
        num=$num-1
        shift
done

wpostdialog sp_cws.node_response node_list "$result"
```

```
exit 0
```

H.1.4 sp_cws.efence_nodes.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.efence_nodes.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog
# INPUT: whitespace-separated list of one or more node numbers
# COMMENTS: Executes the ssp 'Efence' command for all specified
#           node numbers and displays the resulting standard output
#           and standard error in a 'wdispmsg' dialog.
#
#########################################################################

. /etc/Tivoli/setup_env.sh

if [ -z "$*" -o "$*" = "__unchanged__" ]
then
   wdispmsg "You must first select one or more nodes..."
   exit 0
fi

ANSWER=wdispconf "Are you sure you want to fence nodes $* ?"

if [ "$ANSWER" = "NO" ]
then
   exit 0
fi

result=/usr/lpp/ssp/bin/Efence -G $* 2>&1

if [ -n "$result" ]
then
   wdispmsg "$result"
fi

exit 0
```

H.1.5 sp_cws.eunfence_nodes.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.eunfence_nodes.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog
# INPUT: whitespace-separated list of one or more node numbers
# COMMENTS: Executes the ssp 'Eunfence' command for all specified
#           node numbers and displays the resulting standard output
#           and standard error in a 'wdispmsg' dialog.
```

```
#
#########################################################################

. /etc/Tivoli/setup_env.sh

if [ -z "$*" -o "$*" = "__unchanged__" ]
then
   wdispmsg "You must first select one or more nodes..."
   exit 0
fi

ANSWER=wdispconf "Are you sure you want to unfence nodes $* ?"

if [ "$ANSWER" = "NO" ]
then
   exit 0
fi

result=/usr/lpp/ssp/bin/Eunfence -G $* 2>&1

if [ -n "$result" ]
then
   wdispmsg "$result"
fi

exit 0
```

H.1.6 sp_cws.get_all_cw_attributes.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.get_all_cw_attributes.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog
# INPUT: none
# COMMENTS: Obtains names and values of all attributes in the SDR
#           'SP' class and displays them in row/column format.
#
#########################################################################

function doit {
/usr/lpp/ssp/bin/SDRGetObjects SP | awk '

BEGIN {
        firstRow = 1;
}

{
        if (firstRow)
        {
                columns = split($0, key, " ");
                firstRow = 0;
        }
        else
        {
```

```
                split($0, value, " ");
        }
}

END {
        maxKeyLength = 0;
        for (i=1; i<=columns; i++)
        {
           if ((currentKeyLength = length(key[i])) > maxKeyLength)
                        maxKeyLength = currentKeyLength;
        }
        maxKeyLength += 2;
        for (i=1; i<=columns; i++)
        {
                printf("\"%s", key[i]);
                padding = maxKeyLength - length(key[i]);
                for (j=1; j<padding; j++) printf(" ");
                if (value[i] == "\"\"") value[i] = "";
                printf("%s\"{", value[i]);
                printf("\"%s", key[i]);
                for (j=1; j<padding; j++) printf(" ");
                printf("%s\"}", value[i]);
                if (i < columns) printf(",");
        }
}'
}

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

wpostdialog sp_cws.cw_attributes_dialog cw_attributes "doit"

exit $?
```

H.1.7 sp_cws.get_node_numbers.ksh

```
#!/bin/ksh
##############################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.get_node_numbers.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog
# INPUT: none
# COMMENTS: Obtains list of all node numbers (with their respective
#           reliable hostnames) in all system partitions.  The list
#           is formatted to be displayed by an DSL List gadget.
#
##############################################################################

. /etc/Tivoli/setup_env.sh

export PATH=/etc:/tivoli/bin/aix4-r1/CUSTOM:$PATH

nodes=/usr/lpp/ssp/bin/SDRGetObjects -x -G Node node_number reliable_hostname
```

```
set $nodes

typeset -i num=$#

while [[ $num != 0 ]]
do
        temp=echo "$1 $2" | awk '{printf("\"%5s   %s\"{\"%s\"}", $1, $2, $1);}'
        result="$result$temp"
        if [[ $num != 2 ]]
        then
           result=$result","
        fi
        num=$num-2
        shift
        shift
done

echo "$result"
```

H.1.8 sp_cws.launch_applications_driver.ksh

```
#!/bin/ksh
##############################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.launch_applications_driver.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog
# INPUT: name of control workstation that owns the window
# COMMENTS: Posts dialog from which SP applications may be launched.
#
##############################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

wpostdialog sp_cws.sp_applications man_node_name "$1"

exit 0
```

H.1.9 sp_cws.launch_perspectives.ksh

```
#!/bin/ksh
##############################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.launch_perspectives.ksh
# CALLED FROM DIALOG(S): sp_cws.sp_applications
# INPUT PARAMETERS: name of the Perspective application
#                   (vsd, launch pad, etc.)
# COMMENTS: Launches the SP Perspective application indicated by the
#           first argument.  Must first close all file descriptors to
#           be able to return control to the calling window after a
#           Perspectives application has been invoked.
```

```
#
#########################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

export DISPLAY=$WD_DISPLAY

# close stdin, stdout and stderr
exec 0<&-
exec 1<&-
exec 2<&-

/usr/lpp/ssp/bin/$1&

exit 0
```

H.1.10 sp_cws.modify_attribute.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.modify_attribute.ksh
# CALLED FROM DIALOG(S): sp_cws.modify_attribute_dialog
# INPUT: control workstation name, name of attribute being modified,
#        new value for attribute being modified
# COMMENTS: Modifies the specified attribute to the specified new
#           value for the specified control workstation name.
#
#########################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

if [ "$3" = "__unchanged__" ]
then        value=""
else        value="$3"
fi

result=/usr/lpp/ssp/bin/SDRChangeAttrValues SP \
        control_workstation=="$1" "$2"="$value" 2>&1
rc=$?

if [[ $rc != 0 ]]
then
   wdisperr "Failure($rc): $result"
fi

exit $rc
```

H.1.11 sp_cws.modify_attribute_driver.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.modify_attribute_driver.ksh
# CALLED FROM DIALOG(S): sp_cws.cw_attributes_dialog
# INPUT: name of attribute being modified, current value of attribute
#        being modified
# COMMENTS: Posts dialog which accepts new value of attribute being
#           modified.
#
#########################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

echo "$*" >/tmp/BUBJUNK

set $*
name=$1
shift

sp_name=/usr/lpp/ssp/bin/SDRGetObjects SP -x control_workstation

wpostdialog sp_cws.modify_attribute_dialog sp_name \
           "$sp_name" attribute_name "$name" attribute_value "$*"

exit $?
```

H.1.12 sp_cws.power_nodes_off.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.power_nodes_off.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog
# INPUT: whitespace-separate list of one or more node numbers
# COMMENTS: Powers off specified nodes via the ssp 'hmcmds' command.
#           If there is any standard output or standard error, it
#           is displayed in a wdispmsg dialog.
#
#########################################################################

. /etc/Tivoli/setup_env.sh

if [ -z "$*" -o "$*" = "__unchanged__" ]
then
   wdispmsg "You must first select one or more nodes..."
   exit 0
fi
```

```
ANSWER=wdispconf "Are you sure you want to power off nodes $* ?"

if [ "$ANSWER" = "NO" ]
then
   exit 0
fi

typeset -i num=$#
typeset -i frameNumber
typeset -i slotNumber

while [[ $num != 0 ]]
do
        frameNumber=/usr/lpp/ssp/bin/SDRGetObjects -x -G Node \
                    node_number==$1 frame_number
        slotNumber=/usr/lpp/ssp/bin/SDRGetObjects -x -G Node \
                    node_number==$1 slot_number
        allCommands="$allCommands /usr/lpp/ssp/bin/hmcmds -G off \
                    $frameNumber:$slotNumber"
        result=/usr/lpp/ssp/bin/hmcmds -G off \
                    $frameNumber:$slotNumber 2>&1
        if [ -n "$result" ]
        then
           wdispmsg "$result"
        fi
        num=$num-1
        shift
done

wdispmsg "$allCommands"

exit 0
```

H.1.13 sp_cws.power_nodes_on.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.power_nodes_on.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog
# INPUT: whitespace-separate list of one or more node numbers
# COMMENTS: Powers on specified nodes via the ssp 'hmcmds' command.
#           If there is any standard output or standard error, it
#           is displayed in a wdispmsg dialog.
#
#########################################################################

. /etc/Tivoli/setup_env.sh

if [ -z "$*" -o "$*" = "__unchanged__" ]
then
   wdispmsg "You must first select one or more nodes..."
   exit 0
fi
```

```
ANSWER=wdispconf "Are you sure you want to power on nodes $* ?"

if [ "$ANSWER" = "NO" ]
then
   exit 0
fi

typeset -i num=$#
typeset -i frameNumber
typeset -i slotNumber

while [[ $num != 0 ]]
do
        frameNumber=/usr/lpp/ssp/bin/SDRGetObjects -x -G Node \
                    node_number==$1 frame_number
        slotNumber=/usr/lpp/ssp/bin/SDRGetObjects -x -G Node \
                    node_number==$1 slot_number
        allCommands="$allCommands /usr/lpp/ssp/bin/hmcmds -G on \
                    $frameNumber:$slotNumber"
        result=/usr/lpp/ssp/bin/hmcmds -G on \
                    $frameNumber:$slotNumber 2>&1
        if [ -n "$result" ]
        then
           wdispmsg "$result"
        fi
        num=$num-1
        shift
done

wdispmsg "$allCommands"

exit 0
```

H.1.14 sp_cws.run_command_driver_nodes.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.run_command_driver_nodes.ksh
# CALLED FROM DIALOG(S): sp_cws.parent_dialog
# INPUT: whitespace-separated list of one or more node numbers
# COMMENTS: Posts dialog which accepts command (with arguments)
#           to be run on input-specified nodes.
#
#########################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

if [ -z "$*" -o "$*" = "__unchanged__" ]
then
   wdispmsg "You must first select one or more nodes..."
```

```
   exit 0
fi

wpostdialog sp_cws.run_dialog_nodes node_list "\"$*\""

exit $?
```

H.1.15 sp_cws.run_command_nodes.ksh

```
#!/bin/ksh
##############################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.run_command_nodes.ksh
# CALLED FROM DIALOG(S): sp_cws.run_dialog_nodes
# INPUT: whitespace-separated list of nodes and command (with arguments)
#        to run on specified nodes
# COMMENTS: Executes specified command (with arguments) on specified
#           nodes via 'dsh'.  All resulting standard output, standard
#           error, and exit statuses are displayed in a wdispmsg dialog.
#
##############################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

# the following breaks up incoming args so all whitespace is used as separator
# otherwise the list of nodes comes in as ONE argument (instead of many)
set $*

typeset -i num=$#
typeset -i maxNum=$#

while [[ $num != 0 ]]
do
        if [ "$num" = "$maxNum" ]
        then
           nodes=$1
        else
           nodes="$nodes $1"
        fi

        reliableHostname=/usr/lpp/ssp/bin/SDRGetObjects -x -G Node \
                         node_number==$1 reliable_hostname
        if [ "$num" = "$maxNum" ]
        then
           hosts=${reliableHostname%% *}
        else
           hosts=$hosts,${reliableHostname%% *}
        fi

        num=$num-1
        shift
        if [ "$1" = "ArGuMeNtSePaRaToR" ]
        then
```

```
            shift
            break
         fi
done

ANSWER=wdispconf "Are you sure you want to run the command on nodes $nodes ?"

if [ "$ANSWER" = "NO" ]
then
   exit 0
fi

result=/usr/lpp/ssp/bin/dsh -w $hosts \
       "/usr/lpp/ssp/install/bin/spcmdrc $* 2>&1"

wdispmsg "$result"

exit $?
```

H.1.16 sp_cws.spmon.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_cws.spmon.ksh
# CALLED FROM DIALOG(S): sp_cws.sp_applications
# INPUT: none
# COMMENTS: Executes "spmon -g".
#
#########################################################################

export DISPLAY=$WD_DISPLAY

# close stdin, stdout and stderr
exec 0<&-
exec 1<&-
exec 2<&-

/usr/lpp/ssp/bin/spmon -g

exit 0
```

H.1.17 sp_node.get_all_attributes.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_node.get_all_attributes.ksh
# CALLED FROM DIALOG(S): sp_node.parent_dialog
# INPUT: node number
# COMMENTS: Obtains names and values of all attributes in the SDR
#           'Node' class for the specified node number and displays
```

```
#           them in row/column format.
#
#########################################################################

function doit
{
/usr/lpp/ssp/bin/SDRGetObjects -G Node node_number==$1 | awk '

BEGIN {
        firstRow = 1;
}

{
        if (firstRow)
        {
                columns = split($0, key, " ");
                firstRow = 0;
        }
        else
        {
                split($0, value, " ");
        }
}

END {
        maxKeyLength = 0;
        for (i=1; i<=columns; i++)
        {
            if ((currentKeyLength = length(key[i])) > maxKeyLength)
            maxKeyLength = currentKeyLength;
        }
        maxKeyLength += 2;
        for (i=1; i<=columns; i++)
        {
                printf("\"%s", key[i]);
                padding = maxKeyLength - length(key[i]);
                for (j=1; j<padding; j++) printf(" ");
                if (value[i] == "\"\"") value[i] = "";
                printf("%s\"{", value[i]);
                printf("\"%s", key[i]);
                for (j=1; j<padding; j++) printf(" ");
                printf("%s\"}", value[i]);
                if (i < columns) printf(",");
        }
}'
}

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

wpostdialog sp_node.node_attributes_dialog node_attributes "doit $1"

exit $?
```

H.1.18 sp_node.get_frame_number.ksh

```
#!/bin/ksh
#############################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_node.get_frame_number.ksh
# CALLED FROM DIALOG(S): sp_node.parent_dialog
# INPUT: none
# COMMENTS: Writes to standard output the frame number for node on
#           which this script is executing.
#
#############################################################################

ODMDIR=/etc/objrepos
export ODMDIR

NODE=/usr/lpp/ssp/install/bin/node_number
FRAME=/usr/lpp/ssp/bin/SDRGetObjects -x Node \
       node_number==$NODE frame_number

echo "$FRAME"
```

H.1.19 sp_node.get_node_number.ksh

```
#!/bin/ksh
#############################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_node.get_node_number.ksh
# CALLED FROM DIALOG(S): sp_node.parent_dialog
# INPUT: none
# COMMENTS: Writes to standard output the node number for node on
#           which this script is executing.
#
#############################################################################

ODMDIR=/etc/objrepos
export ODMDIR

NODE=/usr/lpp/ssp/install/bin/node_number

echo "$NODE"
```

H.1.20 sp_node.get_slot_number.ksh

```
#!/bin/ksh
#############################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_node.get_slot_number.ksh
```

```
# CALLED FROM DIALOG(S): sp_node.parent_dialog
# INPUT: none
# COMMENTS: Writes to standard output the slot number for node on
#           which this script is executing.
#
#########################################################################

ODMDIR=/etc/objrepos
export ODMDIR

NODE=/usr/lpp/ssp/install/bin/node_number
SLOT=/usr/lpp/ssp/bin/SDRGetObjects -x Node node_number==$NODE slot_number

echo "$SLOT"
```

H.1.21 sp_node.modify_attribute.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_node.modify_attribute.ksh
# CALLED FROM DIALOG(S): sp_node.modify_attribute_dialog
# INPUT: node number, name of attribute to modify, and new value of
#        attribute being modified
# COMMENTS: Modifies the specified attribute value for the specified
#           node to the newly specified value.
#
#########################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

if [ "$3" = "__unchanged__" ]
then
    value=""
else
    value="$3"
fi

result=/usr/lpp/ssp/bin/SDRChangeAttrValues Node \
        node_number=="$1" "$2"="$value" 2>&1
rc=$?

if [[ $rc != 0 ]]
then
    wdisperr "Failure($rc): $result"
fi

exit $rc
```

H.1.22 sp_node.modify_attribute_driver.ksh

```
#!/bin/ksh
#########################################################################
# (C) Copyright IBM Corp. 1997
# All rights reserved.
#
# Sample Tivoli customizations for the SP.
#
# CALLBACK: sp_node.modify_attribute_driver.ksh
# CALLED FROM DIALOG(S): sp_node.node_attributes_dialog
# INPUT: name of attribute being modified, and current attribute value
# COMMENTS: Posts dialog to receive new value for SDR attribute being
#           modified.
#
#########################################################################

export PATH=/tivoli/bin/aix4-r1/CUSTOM:$PATH

. /etc/Tivoli/setup_env.sh

set $*
name=$1
shift

export ODMDIR=/etc/objrepos

node_number=/usr/lpp/ssp/install/bin/node_number

wpostdialog sp_node.modify_attribute_dialog node_number "$node_number"  \
            attribute_name "$name" attribute_value "$*"

exit $?
```

H.2 Examples of AEF Customization Dialogs

```
/*
| NAME: sp_cws.parent_dialog
*/

 Command Dialog
{
    Variables
    {
 CString man_node_name;
 CString man_node_hostid;
 CString man_node_memory;
 CString man_node_os_name;
 CString man_node_os_version;
 CString man_node_os_release;
 CString interface_list;
 CString interpreter;
/*--------------------->>>>> BEGIN SP CUSTOMIZATIONS <<<<<---------------------*/
CString node_numbers = sp_cws.GetNodeNumbers@();
/*---------------------->>>>> END SP CUSTOMIZATIONS <<<<<----------------------*/
    }

    Attributes
```

```
   {
 BitmapTitle = $interpreter;
 Iconic = NO;
 Default = ok;
 HelpMessage = Msg(ManagedNodeHelp,
       "Message catalog not found.
        Consult your system administrator for assistance.
        If the message catalog cannot be found in the NLSPATH,
        contact your Tivoli support provider.",2);
 Name = managed_node_properties;
 PopDown =
     cancel-$owner();
 Purpose = ICONIFY;
/*-------------------->>>>> BEGIN SP CUSTOMIZATIONS <<<<<--------------------*/
Title = Msg(SpManagedNode, "SP Control Workstation", 24);
/*---------------------->>>>> END SP CUSTOMIZATIONS <<<<<----------------------*/
    }

    Gadgets
    {
 Group
 {
     Attributes
     {
         Border = YES;
         Layout = HORIZONTAL;
         Name = g3;
         ChildColumnAlignment = STRETCH;
     }

     Gadgets
     {
         Message
         {
             BitmapTitle = $interpreter;
             Name = managed_node_icon;
             ChildColumnAlignment = LEFT;
             ChildRowAlignment = CENTER;
         }

         Message
         {
             Name = node_name;
/*-------------------->>>>> BEGIN SP CUSTOMIZATIONS <<<<<--------------------*/
Title = Msg(SpManagedNode, "SP Control Workstation:", 25);
/*---------------------->>>>> END SP CUSTOMIZATIONS <<<<<----------------------*/
             Value = $man_node_name;
             ChildColumnAlignment = LEFT;
             ChildRowAlignment = CENTER;
         }

     }
 }

 Group
 {
     Attributes
     {
         Border = YES;
```

```
    Layout = GRID;
    Name = g0;
    Title = Msg(ManagedNodeCatalog,"Properties:",19);
    TitlePos = TOP;
}

Gadgets
{
    Message
    {
        Name = node_name_label;
        Title = Msg(ManagedNodeCatalog,"SystemName:",20);
        GridHorizontal = 2;
        GridVertical = 1;
        ChildColumnAlignment = LEFT;
    }

    Message
    {
        Name = node_name;
        Title = "";
        Value = $man_node_name;
        GridHorizontal = 4;
        GridVertical = 1;
        ChildColumnAlignment = LEFT;
    }

    Message
    {
        Name = host_id_label;
        Title = Msg(ManagedNodeCatalog,"Host ID:",21);
        GridHorizontal = 2;
        GridVertical = 2;
        ChildColumnAlignment = LEFT;
    }

    Message
    {
        Name = host_id;
        Title = "";
        Value = $man_node_hostid;
        GridHorizontal = 4;
        GridVertical = 2;
        ChildColumnAlignment = LEFT;
    }

    Message
    {
        Name = memory_label;
        Title = Msg(ManagedNodeCatalog,"Physical Memory (Mb):",22);
        GridHorizontal = 2;
        GridVertical = 3;
        ChildColumnAlignment = LEFT;
    }

    Message
    {
        Name = memory;
        Title = "";
```

```
    Value = $man_node_memory;
    GridHorizontal = 4;
    GridVertical = 3;
    ChildColumnAlignment = LEFT;
}

Message
{
    Name = os_name_label;
    Title = Msg(ManagedNodeCatalog,
               "Operating System Name:",23);
    GridHorizontal = 2;
    GridVertical = 4;
    ChildColumnAlignment = LEFT;
}

Message
{
    Name = os_name;
    Title = "";
    Value = $man_node_os_name;
    GridHorizontal = 4;
    GridVertical = 4;
    ChildColumnAlignment = LEFT;
}

Message
{
    Name = os_release_label;
    Title = Msg(ManagedNodeCatalog,
               "Operating System Release:",25);
    GridHorizontal = 2;
    GridVertical = 5;
    ChildColumnAlignment = LEFT;
}

Message
{
    Name = os_release;
    Title = "";
    Value = $man_node_os_release;
    GridHorizontal = 4;
    GridVertical = 5;
    ChildColumnAlignment = LEFT;
}

Message
{
    Name = os_version_label;
    Title = Msg(ManagedNodeCatalog,
               "Operating System Version:",24);
    GridHorizontal = 2;
    GridVertical = 6;
    ChildColumnAlignment = LEFT;
}

Message
{
    Name = os_version;
```

```
            Title = "";
            Value = $man_node_os_version;
            GridHorizontal = 4;
            GridVertical = 6;
            ChildColumnAlignment = LEFT;
        }

    }
}

/*-------------------->>>>> BEGIN SP CUSTOMIZATIONS <<<<<--------------------*/
Message
{
 Name = unnamed-GADGET-1-SP;
 Title = "";
}

Group
{
 Attributes
 {
  Name = g_sp;
  Title = Msg(SpManagedNode, "SP Properties:", 26);
  TitlePos = TOP;
  Border = YES;
  Layout = VERTICAL;
 }

 Gadgets
 {
  Group
  {
   Attributes
   {
    Name = g4;
    Title = Msg(SpManagedNode, "Node Control:", 27);
    TitlePos = TOP;
    Border = YES;
    Layout = HORIZONTAL;
   }

   Gadgets
   {
    List
    {
     Name = node_list;
     Border = NO;
     Choices = $node_numbers;
     Columns = 5;
     Rows = 6;
     ReadOnly = NO;
     Show = SOME;
     Sort = YES;
     ShowBrowser = YES;
    }

    Group
    {
     Attributes
```

```
{
 Name = power_buttons;
 Alignment = LEFT;
 Layout = VERTICAL;
}

Gadgets
{
 Button
 {
  Name = check_node_response;
  Title = Msg(SpManagedNode, "Response", 28);
  Commands = sp_cws.CheckNodeResponse@($g_sp.g4.node_list);
  HelpMessage = Msg(ManagedNodeHelp,
                  "check node response of selected
                   nodes of the adjacent list",2);
 }

 Button
 {
  Name = run_command_driver_nodes;
  Title = Msg(SpManagedNode, "Run", 29);
  Commands = sp_cws.RunCommandDriverNodes@($g_sp.g4.node_list);
  HelpMessage = Msg(ManagedNodeHelp,
     "run a command on selected nodes on the adjacent list",2);
 }

 Button
 {
  Name = power_node_on;
  Title = Msg(SpManagedNode, "Power On", 30);
  Commands = sp_cws.PowerNodesOn@($g_sp.g4.node_list);
  HelpMessage = Msg(ManagedNodeHelp,
               "power on selected nodes on the adjacent list",2);
 }

 Button
 {
  Name = power_node_off;
  Title = Msg(SpManagedNode, "Power Off", 31);
  Commands = sp_cws.PowerNodesOff@($g_sp.g4.node_list);
  HelpMessage = Msg(ManagedNodeHelp,
              "power off selected nodes on the adjacent list",2);
 }

 Button
 {
  Name = efence;
  Title = Msg(SpManagedNode, "Efence", 32);
  Commands = sp_cws.EfenceNodes@($g_sp.g4.node_list);
  HelpMessage = Msg(ManagedNodeHelp,
               "fence selected nodes on the adjacent list",2);
 }

 Button
 {
  Name = eunfence;
  Title = Msg(SpManagedNode, "Eunfence", 33);
  Commands = sp_cws.EunfenceNodes@($g_sp.g4.node_list);
```

```
        HelpMessage = Msg(ManagedNodeHelp,
                     "unfence selected nodes on the adjacent list",2);
       }
      }
     }
    }
   }

   Message
   {
    Name = unnamed-GADGET-2-SP;
    Title = "";
   }

   Group
   {
    Attributes
    {
     Name = attribute_buttons;
     Alignment = LEFT;
     Layout = HORIZONTAL;
    }

    Gadgets
    {
     Button
     {
      Name = cw_attributes;
      Title = Msg(SpManagedNode, "All Attributes", 34);
/*      Commands = sp_cws.GetAllCwAttributes@();  */
      Commands = sp_cws.GetAllCwAttributes@();
      HelpMessage = Msg(ManagedNodeHelp,
                   "Read/Write all SDR information about
                    this control workstation",2);
     }

     Button
     {
      Name = run_command_driver;
      Title = Msg(SpManagedNode, "Run CW", 35);
      Commands = sp.RunCommandDriver@();
      HelpMessage = Msg(ManagedNodeHelp,
         "Command line interface to the control workstation",2);
     }

     Button
     {
      Name = launch_applications_driver;
      Title = Msg(SpManagedNode, "SP Applications", 36);
      Commands = sp_cws.LaunchApplicationsDriver@($man_node_name);
      HelpMessage = Msg(ManagedNodeHelp,
         "Launch SP applications (Perspectives and System Monitor)",2);
     }
    }
   }
  }
 }
/*---------------------->>>>> END SP CUSTOMIZATIONS <<<<<----------------------*/
```

```
Message
{
    Name = unnamed-GADGET-2;
    Title = "";
}

Group
{
    Attributes
    {
        Border = YES;
        Layout = HORIZONTAL;
        Name = g2;
        Title = Msg(ManagedNodeCatalog,"IP Interfaces:",26);
        TitlePos = TOP;
    }

    Gadgets
    {
        Choice
        {
            Border = NO;
            Choices = $interface_list;
            Name = ip_list;
            ShowBrowser = NO;
        }

        Group
        {
            Attributes
            {
                Alignment = LEFT;
                Layout = VERTICAL;
                Name = ip_buttons;
                ChildRowAlignment = STRETCH;
            }

            Gadgets
            {
                Button
                {
                    Commands =
                        add_interface—$owner();
                    Name = add_interface;
                    Title = Msg(ManagedNodeCatalog,
                                "Add Interface...",27);
                    ChildColumnAlignment = STRETCH;
                }

                Button
                {
                    Commands =
                        remove_interface—$owner($g2.ip_list);
                    Name = remove_interface;
                    Title = Msg(ManagedNodeCatalog,
                                "Remove Interface",28);
                    ChildColumnAlignment = STRETCH;
                }
```

```
                Button
                {
                    Commands =
                        edit_interface—$owner($g2.ip_list);
                    Name = edit_interface;
                    Title = Msg(ManagedNodeCatalog,
                               "Edit Interface",29);
                    ChildColumnAlignment = STRETCH;
                }

                Button
                {
                    Commands =
                        reset—$owner();
                    Name = reset_interfaces;
                    Title = Msg(ManagedNodeCatalog,Reset,30);
                    ChildColumnAlignment = STRETCH;
                }

            }
        }

    }
}

Message
{
    Name = unnamed-GADGET-4;
    Title = "";
}

CommandButton
{
    Commands =
        ok—$owner();
    Name = ok;
    Title = Msg(ManagedNodeCatalog,"Update & Close",14);
}

CommandButton
{
    Commands =
        cancel—$owner();
    Name = cancel;
    Title = Msg(GenericCollectCat,Close,40);
}

   }
}
/*=============================================================================
| Dialog which launches various pre-existing SP applications.
| NAME: sp_cws.sp_applications
=============================================================================*/

Command Dialog
{
 Variables
 {
  CString man_node_name;
```

```
}

Attributes
{
 Name = sp_launch_applications;
 Title = $man_node_name;
}

Group
{
 Attributes
 {
  Name = g1;
  Title = Msg(SpManagedNode, "SP Perspectives", 1);
  TitlePos = TOP;
  Border = YES;
  Layout = VERTICAL;
  ChildColumnAlignment = STRETCH;
 }

 Gadgets
 {
  Button
  {
   Name = launch_pad;
   Title = Msg(SpManagedNode, "Launch Pad", 2);
   Commands = sp_cws.LaunchPerspectives@("perspectives");
   HelpMessage = Msg(SpManagedNode,
                "Start the SP Perspectives Launch Pad", 3);
  }

  Button
  {
   Name = hardware;
   Title = Msg(SpManagedNode, "Hardware", 4);
   Commands = sp_cws.LaunchPerspectives@("sphardware");
   HelpMessage = Msg(SpManagedNode,
                "Launch the SP Perspectives Hardware view", 5);
  }

  Button
  {
   Name = event;
   Title = Msg(SpManagedNode, "Event", 6);
   Commands = sp_cws.LaunchPerspectives@("spevent");
   HelpMessage = Msg(SpManagedNode,
                "Launch the SP Perspectives Event view", 7);
  }

  Button
  {
   Name = vsd;
   Title = Msg(SpManagedNode, "VSD", 8);
   Commands = sp_cws.LaunchPerspectives@("spvsd");
   HelpMessage = Msg(SpManagedNode,
         "Launch the SP Perspectives Virtual Share Disk view", 9);
  }

  Button
```

```
  {
   Name = syspar;
   Title = Msg(SpManagedNode, "System Partitioning Aid", 10);
   Commands = sp_cws.LaunchPerspectives@("spsyspar");
   HelpMessage = Msg(SpManagedNode,
      "Launch the SP Perspectives System Partitioning Aid view", 11);
  }

  Button
  {
   Name = perfmon;
   Title = Msg(SpManagedNode, "System Performance Tool", 12);
   Commands = sp_cws.LaunchPerspectives@("spperfmon");
   HelpMessage = Msg(SpManagedNode,
      "Launch the SP Perspectives System Performance Tool view", 13);
  }
 }
}

Group
{
 Attributes
 {
  Name = g2;
  Title = Msg(SpManagedNode, "SP System Monitor", 14);
  Border = YES;
  Layout = VERTICAL;
  ChildColumnAlignment = STRETCH;
  TitlePos = TOP;
 }

 Gadgets
 {
  Button
  {
   Name = system_monitor;
   Title = Msg(SpManagedNode, "System Monitor GUI", 15);
   Commands = sp_cws.spmon@();
   HelpMessage = Msg(SpManagedNode,
                "Launch the SP Hardware Monitor GUI", 16);
   TitlePos = TOP;
  }
 }
}

Group
{
 Button
 {
  Name = dismiss;
  Title = Msg(GenericCollectCat,Dismiss,40);
  Commands = dtc_dismiss($owner,sp_launch_applications, $self);
 }
}
}
```

Appendix I. Special Notices

This publication is intended to help customers, business partners and IBM personnel to integrate the RS/6000 SP-specific system management data into a Tivoli management environment. The information in this publication is not intended as the specification of any programming interfaces that are provided by Tivoli Management Environment 10 and Parallel System Support Program. See the PUBLICATIONS section of the IBM Programming Announcement for Tivoli Management Environment 10 and Parallel System Support Program for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

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

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

AIX	AIX/6000
AS/400	BookManager
IBM	NetView
POWERparallel	RS/6000
Scalable POWERparallel Systems	SP
System/390	SystemView

The following terms are trademarks of other companies:

Tivoli, Tivoli/Enterprise Console, TME 10 are trademarks of Tivoli Systems, an IBM Company.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

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

Appendix J. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

J.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 233.

- *RS/6000 SP High Availability Infrastructure*, SG24-4838
- *RS/6000 SP: Problem Determination Guide*, SG24-4778
- *RS/6000 SP Monitoring: Keeping It Alive*, SG24-4873
- *Understanding Tivoli's TME 3.0 and TME 10*, SG24-4948
- *TME 10 Cookbook for AIX Systems Management and Networking Applications*, SG24-4867

J.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

J.3 Other Publications

These publications are also relevant as further information sources:

- *IBM Parallel System Support Programs for AIX: Administration Guide*, GC23-3897
- *IBM Parallel System Support Programs for AIX: Command and Technical Reference*, GC23-3900
- *IBM Parallel System Support Programs for AIX: Event Management Programming Guide and Reference*, SC23-3996
- *IBM Parallel System Support Programs for AIX*, SBOF-8587
- *Tivoli Courier User's Guide*, GC31-8330
- *TME 10 Inventory User's Guide*, GC31-8381
- *Tivoli Enterprise Console Event Integration Facility Guide*, GC31-8337
- *Tivoli AEF User's Guide*, GC31-8345

These publications are distributed with the TME 10 products:

- *TME 10 Framework Reference Manual*, SC31-8434
- *TME 10 Framework Planning and Installation Guide*, SC31-8432
- *TME 10 Framework Release Notes Version 3.1, Revision C*, GI10-3082
- *TME 10 Framework User's Guide*, GC31-8433
- *TME 10 User Administration Installation Guide*, SC31-8389
- *Tivoli User Administration User and Group Management Guide*, SC31-8391
- *TME 10 Task Library Language Developer's Guide*, SC31-8436
- *Tivoli Sentry User's Guide*, GC31-8382
- *Tivoli Enterprise Console User's Guide Volume I*, GC31-8334
- *Tivoli Enterprise Console User's Guide Volume II*, GC31-8335
- *Tivoli Enterprise Console Rule Builder's Guide*, GC31-8336
- *Tivoli Enterprise Console Event Adapter Guide: SNMP*, SC31-8342
- *Tivoli Enterprise Console Event Adapter Guide: LogFile*, SC31-8343
- *Tivoli Enterprise Console Event Adapter Guide: IBM NetView*, SC31-8339

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO: type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Web Site on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

ITSO Redbook Evaluation

Integrating TME 10 on the RS/6000 SP
SG24-2071-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

_____

_____

_____

_____

_____

List of Abbreviations

ADE	TME 10 Application Developers Environment
AEF	TME 10 Application Extension Facility
AIX	Advanced Interactive Executive
API	Application Programming Interface
BAROC	Basic Representation of Object in C
EIF	TME 10 Event Integration Facility
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
PSSP	Parallel Systems Support Program
SBS	Structured Byte String
SDR	System Data Repository
SMIT	System Management Interface Tool
SNMP	Simple Network Management Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TME	Tivoli Management Environment

Index

Special Characters

A

B

C

D

E

F

G

I

J

K

L

M

W

Printed in U.S.A.