Tivoli

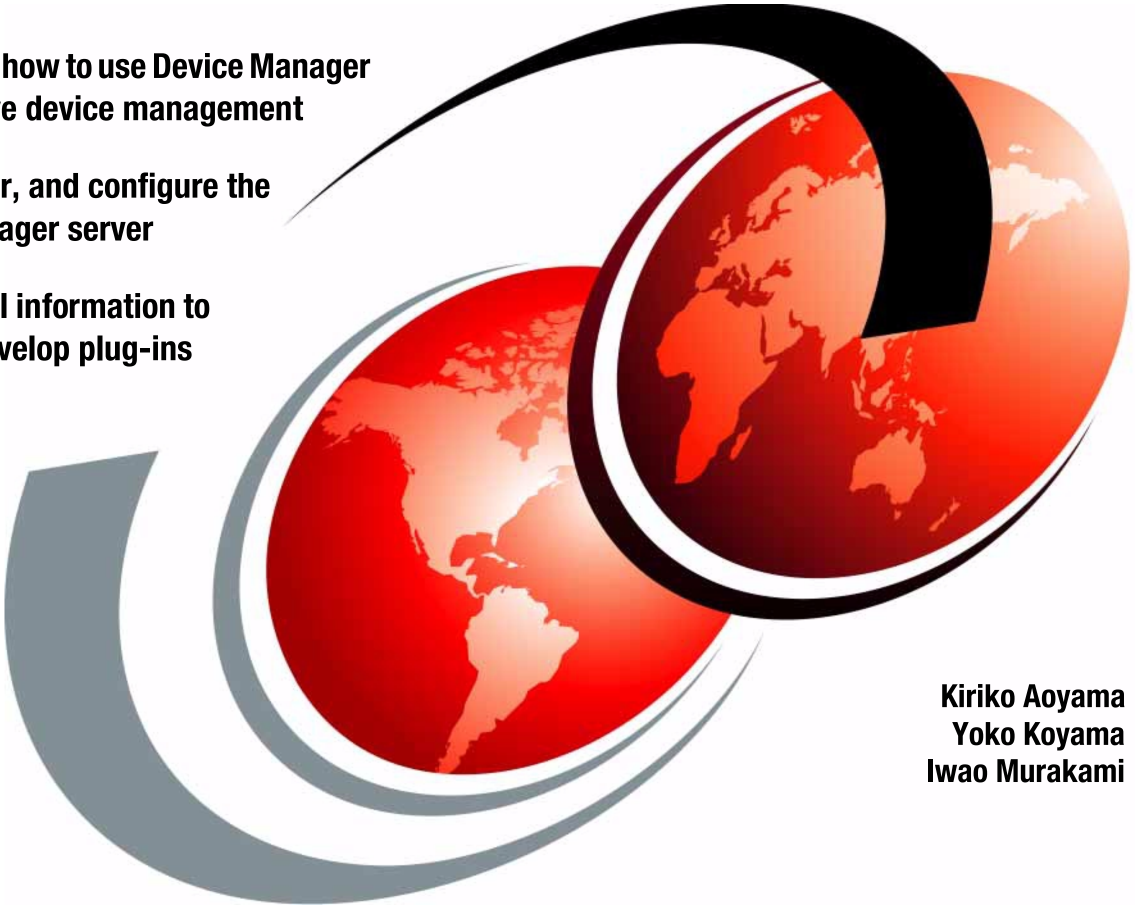
Tivoli Personalized Services Manager Device Manager 1.1

Pervasive Device Management

Understand how to use Device Manager for pervasive device management

Install, tailor, and configure the Device Manager server

Learn useful information to help you develop plug-ins



Kiriko Aoyama
Yoko Koyama
Iwao Murakami

Redbooks

ibm.com/redbooks

SG24-6027-00

International Technical Support Organization

Tivoli Personalized Services Manager Device Manager 1.1: Pervasive Device Management

February 2001

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special notices” on page 255.

First Edition (February 2001)

This edition applies to Version 1.1 of the Device Manager that is included in Tivoli Personalized Services Manager Version 1.1, 5698-PSM, IBM WebSphere Everyplace Suite Enterprise Edition Version 1.1, 5765-E59, and IBM WebSphere Everyplace Suite Service Provider Edition Version 1.1, 5697-G53.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. OSJB Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

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

Device Manager Version 1.1 is software that functions as a part of Tivoli Personalized Services Manager Version 1.1, or IBM WebSphere Everyplace Suite Version 1.1. It enables pervasive devices to easily access a set of network-based services, as well as Internet-based e-commerce services.

This IBM Redbook will assist developers and system architects who are involved in building pervasive device management solutions that use Device Manager Version 1.1. It explains how Device Manager Version 1.1 can be used to manage devices in the growing pervasive computing world. Pervasive devices are typically small, resource-limited, and not perceived as computers; however, growth of these information appliances is increasing at an amazing rate.

In this redbook, you will find information that to help you plan, tailor, and configure the Device Manager to successfully implement solutions that an e-business must address to access Internet-based services from pervasive devices such as Palm devices, screenphones, and Wireless Application Protocol (WAP) devices.

This redbook also looks at software distribution and device configuration examples, introduction of supported APIs, and an explanation of device plug-ins. It helps you to plan and make your own device plug-in when you want to manage special devices not provided with Device Manager Version 1.1.

Prior to reading this redbook, you should have a basic knowledge of pervasive computing, as well as some understanding of Web and Java technologies, and the terminology used in the Internet and pervasive computing world.

The team that wrote this redbook

This redbook was produced as a result of a special ITSO residency project conducted by the Pervasive Computing (PvC) division of IBM Japan's Systems Engineering Co., Ltd (ISE). Three specialists were assigned to this project to develop this redbook.

Kiriko Aoyama is an I/T specialist at ISE. She has ten years of experience in the AIX field. She has been an IBM Certified Advanced Technical Expert - RS/6000 AIX since 1998. Her areas of expertise include a wide range of AIX related products, particularly RS/6000 SP systems. She currently focuses in the area of pervasive computing.

Yoko Koyama is an I/T Engineer in ISE. This company provides technical support to customers and IBM Japan for all IBM hardware and software products. She has three years of experience in technical support in her organization. She was a member of the TPSM Device Manager Plug-ins 1.1 FVT team, and was involved in the testing of the Palm plug-in.

Iwao Murakami is an advisory I/T specialist in ISE. He conducted this redbook project as a project leader in Japan. He worked as a 3270 developer in Yamato Laboratory for ten years. Since 1995, he has been a group leader of an ISE technical support group for PC areas such as host integration (Pcomm, Hod, and so on), Netfinity, Warp and Windows. He currently focuses on driving new business and taking advantage of the pervasive business opportunities in Japan.

ISE is a wholly owned subsidiary of IBM Japan and is responsible for providing technical support of IBM products to customers directly, as well as to IBM Japan. ISE provides technical support for a wide range of products, not only IBM products, but other major software such as Microsoft or Oracle products. See the ISE Web site at:
`http://www.jp.ibm.com/ise/english/ecomp.htm`

This redbook could not be completed without significant help from Yamato Software Development Laboratory (YSL), Software Group.

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

Kenji Kawasaki
Manager of PvC WES Project Management, YSL, SWG

Shuhzoh Kusuda
Project Manager of Device Management Development, YSL, SWG

Kazuhito Akiyama
Software Engineer, Device Management Development, YSL, SWG

Hiroaki Kameda
Project Manager of WES R1 Development, YSL, SWG

Hiroshi Suzuki
Staff Software Engineer, WES R1 Development, YSL, SWG

Theresa Morris
Kent Hayes Jr
Dean Skidmore
Tivoli Systems

Stephen Hochstetler
International Technical Support Organization, Austin Center

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 “IBM Redbooks review” on page 283 to the fax number shown on the form.
- Use the online evaluation form found at **ibm.com**/redbooks
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. Introduction to Device Manager

This chapter provides the information you need to begin understanding Device Manager. It contains information about Device Manager software components and supported devices.

We also discuss the functionality to extend e-business applications to the new classes of information applications, for example, Wireless Application Protocol (WAP) devices, personal digital assistants (PDAs), and screenphones.

1.1 What is Device Manager

Device Manager is software that helps service providers manage their subscribers' devices. In this book, the term *devices* means information appliances, such as the personal digital assistants (PDAs), handheld PCs, sub-notebooks, smartphones, Wireless Application Protocol (WAP) devices, set-top boxes, in-vehicle information systems, and other emerging devices for pervasive computing.

Typically small, resource-limited, and not perceived as computers, these information appliances are increasingly used to access that spectrum of network-based services, including Internet-based e-commerce services. Pervasive devices give their users, both expert and novice, the ability to conveniently access and take action on information. Users can surf the Net, send and receive e-mail, shop and bank online, and even arrange remote home networking, often through a service provider.

On the other hand, people often experience frustration when setting up their devices, updating and reconfiguring software, or subscribing to new services. Even experienced users may have little knowledge about how their own devices really work.

To offer the most value, and often to secure brand loyalty, service providers need to help consumers efficiently manage their information and devices. Device Manager addresses these needs, to include management of devices and their related resources, like applications or rest pages.

As the popularity of pervasive devices grows steadily, service providers are further challenged to manage thousands or even millions of devices effectively. Device Manager will help service providers offer value-added service to their customers by managing their information and devices, even when they have millions of devices.

You use the Device Manager as a part of Tivoli Personalized Services Manager 1.1 (5698-PSM). The Tivoli Personalized Services Manager (TPSM) provides a completely integrated infrastructure of software components for Internet service provisioning.

For more information about Tivoli Personalized Services Manager, refer to 1.1.3, "DM in Tivoli Personalized Services Manager (TPSM)" on page 6.

You can use TPSM as a component of IBM WebSphere Everyplace Suite Enterprise Edition 1.1 (5765-E59), or IBM WebSphere Everyplace Suite Service Provider Edition 1.1 (5697-G53). IBM WebSphere Everyplace Suite is an integrated, modular suite of software components. These software components provide reliable access to online information from a wide variety of pervasive devices.

For more information about IBM WebSphere Everyplace Suite, refer to 1.1.4, "TPSM in WebSphere Everyplace Suite (WES)" on page 8.

1.1.1 Overview

The purpose of Device Manager is to extend the functionality of the required Subscription Manager product to include management of devices and their related resources, like device application software.

For example, by using the software distribution function, the service provider can automatically send the latest information and updated program whenever their subscribers connect to their server. Subscribers do not need to pay attention to their program versions, as long as their devices are managed by a Device Manager server.

Device Manager must work in conjunction with Tivoli Personalized Services Manager. Therefore, Figure 1 shows the components supplied with Device Manager and the required Subscription Manager product and its relationship to those components.

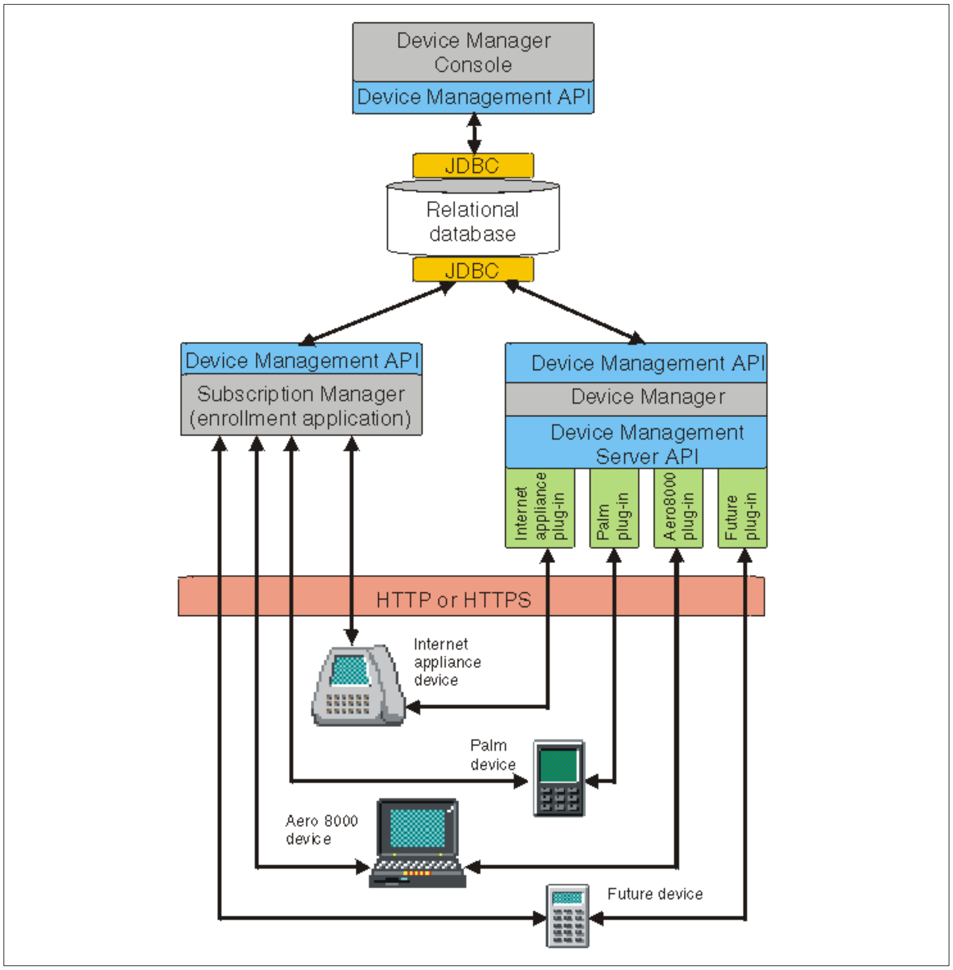


Figure 1. Device Manager overview

Figure 1 also shows the required Device Manager relational database. All job and device information is stored in tables in this database. The relational database is a required product and not a component, but it is shown to illustrate its relationship to the components. As shown in the figure, the Subscription Manager product and Device Manager share the same database.

Device Manager uses Java technology. Its servers run on IBM AIX operating system or Sun Solaris environments. It uses a global, relational database for data storage. DB2 Universal Database Version 7 and Oracle 8 are supported with DMS 1.1. Its administration clients run in Microsoft Windows operating system environments, where the Device Manager console for administrators is deployed.

Device Manager can be installed as a single-server deployment or, for larger user and device numbers, as a multiserver environment with a network dispatcher as the front end. Using a Network Dispatcher as a Load Balancer, you can keep scalability and high availability for a number of Device Manager servers.

1.1.2 Device Manager components

In this section, we briefly describe the functions of each Device Manager component. For details about the functions and behaviors of each component, refer to 2.2, “Components of Device Manager” on page 30.

Device Manager consists of six components:

- Device Manager server
- Device plug-ins
- Device management server API
- Device Manager database
- Device management API
- Device Manager console

1.1.2.1 Device Manager server

A Device Manager server is a computer that includes the device management server servlet (DMS servlet) and the device plug-ins. When a new job is submitted, Device Manager server processes jobs for devices.

The DMS servlet and device plug-ins work together to process a job. When a device connects to the service provider's network, it is directed to a Device Manager server for job processing. The device is redirected to an enrollment application if it is not currently enrolled with Device Manager.

1.1.2.2 Device plug-ins

A device plug-in resides on the Device Manager server and provides the logic that handles device identification, communications, job processing, and high-level management tasks for a particular class of managed devices.

A device class is made up of specific kinds of manufacturer devices whose operations can be similarly managed. For example, one device class provided with Device Manager includes Palm PDAs that run PalmOS 3.0.1 (or later). Another device class might include Internet appliance devices that use the Windows CE operating system and a specific CPU. Device plug-ins are typically developed and provided with Device Manager, but they can also originate with a device manufacturer or integrator, and be installed later on a Device Manager server.

1.1.2.3 Device management server API

The device management server API (package com.tivoli.dms.dmserver) defines the programming interface between the device management server servlet (DMS servlet) and the device plug-ins. This interface allows device plug-ins to interact with a DMS servlet.

1.1.2.4 Device Manager database

The Device Manager database is the repository for all device management information. It is implemented in a relational database and accessed using the device management API. The database contains tables of entries that describe devices and device-related data resources.

1.1.2.5 Device management API

The device management API (package com.tivoli.dms.dmapi) defines the programming interface between Device Manager servers, administration clients, or external applications, and the device-related data resources stored in the Device Manager database.

Resources that can be managed include table entries describing devices and classes of devices, their parameters, the device class template, software defined for and available to devices, software actually installed on devices by Device Manager, Device Manager servers, and jobs and classes of jobs. Methods are provided to add, delete, and modify these entries, and to search the database for entries matching certain criteria.

For more information about table contents, refer to 5.1, “Data model” on page 146.

1.1.2.6 Device Manager console

The Device Manager console is a graphical user interface (GUI) for administering device management operations from a Microsoft Windows client. Administrators use this interface to complete tasks such as add and view devices and device software, submit jobs for devices, and query job status.

The console invokes methods of the device management API to access information in the Device Manager database and perform requested operations. External applications, such as enrollment and billing, can also use this API to retrieve device information from the database and submit Device Management jobs.

1.1.3 DM in Tivoli Personalized Services Manager (TPSM)

The Tivoli Personalized Services Manager provides a completely integrated infrastructure of software components for Internet service provisioning. It is an industrial strength system designed to allow the Internet Service Provider (ISP) to support several separately branded offerings simultaneously and to provide each brand with a unique marketplace identity and with a full range of business offerings.

The Tivoli Personalized Services Manager is designed for continuous operation, flexible enhancement and high scalability. The system relies on industry standard hardware and software components with a flexible architecture for integrating additional and varied components.

ISPs using the Tivoli Personalized Services Manager can offer their subscribers an unlimited variety of Internet features without having to alter a core base of information management and customer service features.

Figure 2 shows Tivoli Personalized Services Manager (TPSM) components.

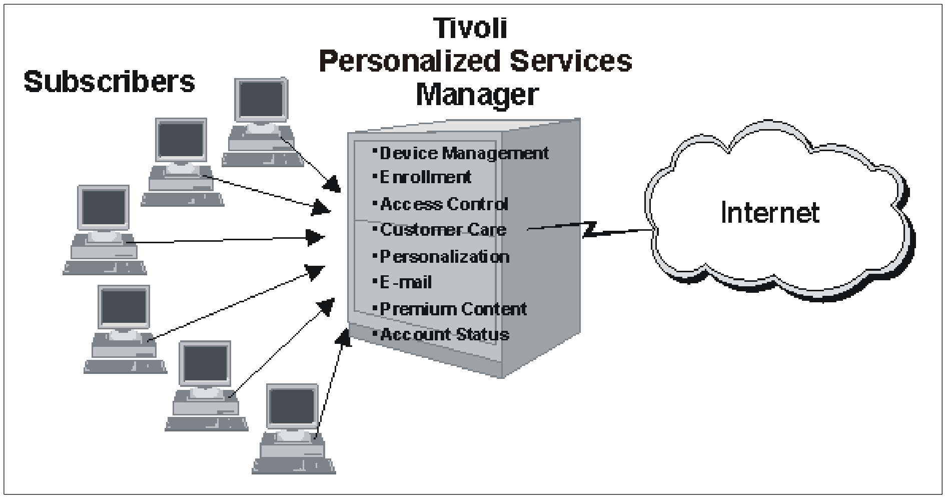


Figure 2. Tivoli Personalized Services Manager

Device Manager is the component of Tivoli Personalized Services Manager responsible for the management of devices.

In the remainder of this section, we briefly introduce the following pvTPSM components to better understand the relationships between each component:

- Enrollment and service provisioning
- Subscriber authentication and access control
- Customer care support
- Personalization
- Self-care

If you want information about all of the TPSM components, refer to the online manual *Tivoli Internet and Personalized Services Manager: Planning and Installation*.

1.1.3.1 Enrollment and service provisioning

ISPs use a centrally managed enrollment engine while being allowed autonomy in presentation and payment plan offers. Using the enrollment and service provision function of TPSM, a standard set of enrollment screens can be customized to deliver uniquely branded messages and graphics, as well as ISP-specific billing plans and payment options.

Behind the scenes, a consistent array of data elements is captured from each new subscriber, thereby distributing all of Tivoli Personalized Services Manager's additional features to every subscriber, regardless of their ISP. With individual branding comes unique realm name distinction.

Each ISP hosted on Tivoli Personalized Services Manager is assigned a realm, which is a partition within the overall database that distinguishes them and their subscribers from all others in the system.

Realms help to extend brand name identification. Realm A could be “ALPHA.com” and Realm B could be “BETA.com”. Every subscriber to the ISP named “ALPHA” would receive a user name and e-mail address that is unique to the realm of “ALPHA.com” (for example, JOHN@ALPHA.com). Every subscriber enrolled with the ISP named “BETA” would get a user name and e-mail address unique to “BETA.com”.

Realm distinctions provide benefits to system management. Customer Care Representatives (CSRs) can be granted read and write security provisions to specific realms. This keeps the privacy of a realm's data separate and secure.

1.1.3.2 Subscriber authentication and access control

Subscribers are granted single sign-on status throughout a session. Information pertinent to the continuation of the session is preserved using cookies, eliminating the need to re-enter ID information. For access to critically secure data, Tivoli's Secureway Policy Director can be used in conjunction with the Tivoli Personalized Services Manager.

1.1.3.3 Customer Care support

All data viewed by CSRs is completely up-to-date and any changes are immediately accessible to the billing system and any other integrated systems. Access control for customer service personnel can be restricted in several ways. For example, they can be limited to read or write capabilities, or they can be allowed to only access certain portions of the subscriber population.

This degree of security control allows ISPs within TPSM to subcontract customer service help in off-site locations where CSRs can log in to the Tivoli Personalized Services Manager Customer Care application remotely and securely work with subscriber data.

1.1.3.4 Personalization

Personalization allows users to customize their home page. Subscribers can actively customize their portal page by configuring categories such as:

- Stock quotes (personal portfolio)
- Weather (local, national, or international)
- Entertainment (music, movies, theater, and so forth)
- Travel (destinations, prices, and so forth)
- News (local, national, or international)

The Tivoli Personalized Service Manager and virtual ISPs can personalize subscriber home pages over and above the choices made by subscribers by adding:

- Links to local content based on subscriber profile
- Targeted ads pulled from an external ad-queue management system

1.1.3.5 Self-care

Self-care is an another aspect of personalization. Subscribers can access and modify some of their profile data. They can update their address and telephone data, change their billing plan and method of payment data, and subscribe to premium content through the Tivoli Personalized Services Manager's self-care application. This application is linked to the central database from links on their portal page.

1.1.4 TPSM in WebSphere Everyplace Suite (WES)

You can use the Tivoli Personalized Services Manager as a stand-alone system or as a component of the WebSphere Everyplace Suite. The IBM WebSphere Everyplace Suite is an integrated, modular suite of software components.

These software components provide reliable access to online information from a wide variety of pervasive devices such as cellular phones, personal digital assistants (PDAs), and mobile computers, among other wireless and traditionally connected devices. Each IBM WebSphere Everyplace Suite component performs a different function in extending pervasive computing connectivity.

If you use IBM WebSphere Everyplace Suite, you can install all components using the same interface. The installation wizard prompts you for all the necessary information to complete the installation. For all of the components in the Everyplace Suite, the installation wizard automatically uses that component's installation program to properly install the component. The Everyplace Suite installation wizard also uses the information to provide limited configuration of each component after it is installed.

For detailed information about installation instructions, refer to the online manual *WebSphere Everyplace Suite Getting Started v1.1*.

Figure 3 shows the Everyplace Suite providing connectivity between client software on the pervasive devices and Internet applications and content.

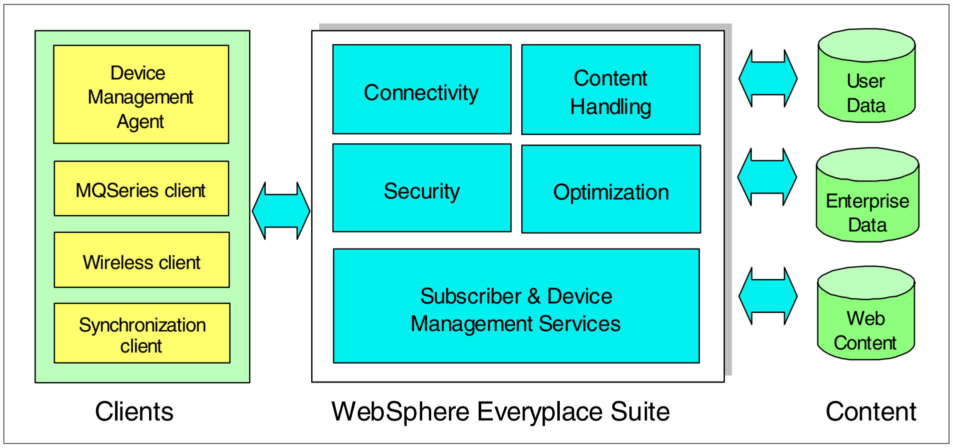


Figure 3. WebSphere Everyplace Suite overview

The Everyplace Suite components provide the following services through the corresponding components:

- **Connectivity**
  - *Everyplace Wireless Gateway*: Provides a communications platform that enables Internet Protocol and Wireless Application Protocol (WAP) applications to run in a wireless environment.

  - *Everyplace Authentication Server*: Acts as the point of entry to the Everyplace Suite domain for devices that do not connect through the Everyplace Wireless Gateway.
  - *MQSeries Everyplace*: Provides assured messaging capability between devices and any MQSeries family platform.
  - *Everyplace Synchronization Manager*: Enables mobile computing devices to link remotely to applications such as Microsoft Exchange, Lotus Notes or DB2 databases.
- **Security**
  - *Everyplace Wireless Gateway*: Provides network access user authentication for WAP and non-WAP users, and data encryption. Supports Internet Protocol and WAP transport layer security and can be configured to use a third party RADIUS server.
  - *Everyplace Authentication Server*: Authenticates users defined to the Everyplace Suite when they attempt to access Everyplace Suite services.
- **Subscriber and device management**
  - *Tivoli Personalized Services Manager*: Provides tools to centrally manage subscribers and their devices, and allows for the creation of discrete groups of users.
- **Content handling**
  - *WebSphere Transcoding Publisher*: Adapts, reformats, and filters data based on the destination device or network.
  - *WebSphere Edge Server Caching Proxy (Web Traffic Express)*: Retrieves Internet data for multiple browser clients and acts as a caching server and content filter.
- **Optimization**
  - *WebSphere Edge Server Load Balancer (Network Dispatcher)*: Balances requests in real time among Everyplace Suite servers to increase capacity and scalability of heavily accessed enterprises.

Figure 4 shows the data path in the IBM WebSphere Everyplace Suite environment.

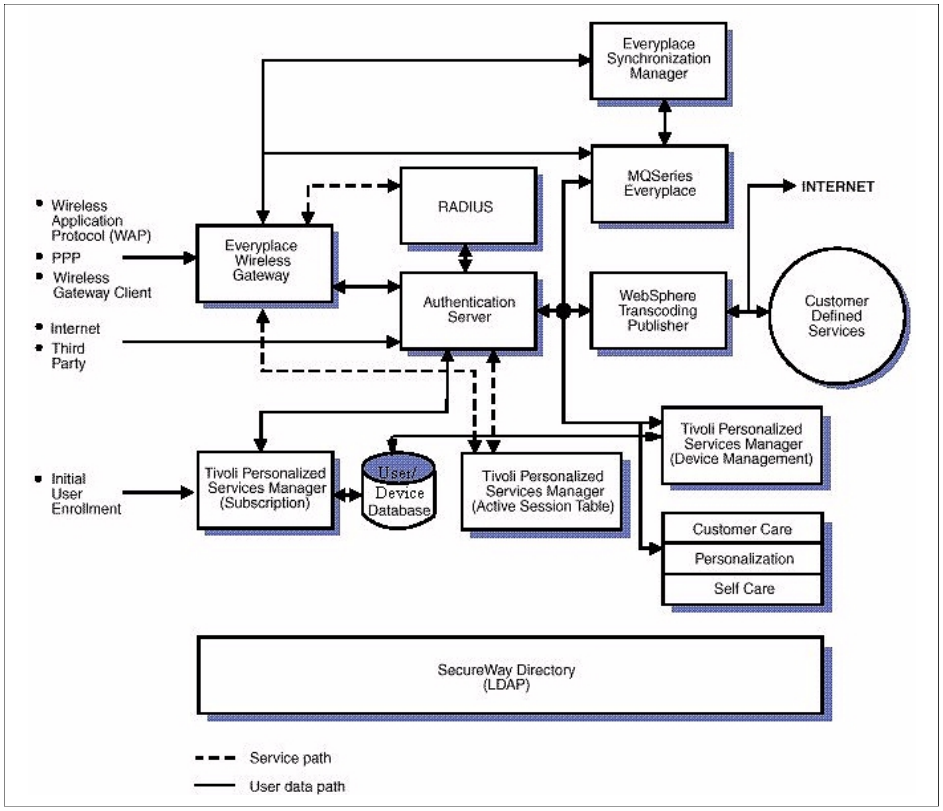

Everyplace Synchronization Manager
MQSeries Everyplace
INTERNET
RADIUS
Wireless Application Protocol (WAP)
PPP
Wireless Gateway Client
Everyplace Wireless Gateway
Authentication Server
WebSphere Transcoding Publisher
Customer Defined Services
Internet
Third Party
Tivoli Personalized Services Manager (Device Management)
Initial User Enrollment
Tivoli Personalized Services Manager (Subscription)
User/ Device Database
Tivoli Personalized Services Manager (Active Session Table)
Customer Care
Personalization
Self Care
SecureWay Directory (LDAP)
Service path
User data path

Figure 4. WebSphere Everyplace Suite components

1.1.5 Device Manager functions

The Device Manager feature provides a flexible framework and a set of services for managing subscriber's devices including:

- Enrollment

  Easy methods to enroll new subscribers and their devices, and enroll devices for existing subscribers.

- Update device configuration remotely

  Perform initial device setup and change device setup as needed.

- Distribute software to the device

  The service provider can centrally manage software and configure Tivoli Personalized Services Manager to automatically distribute software to subscriber's personal devices.

- Update rest pages (startup pages) for devices supporting this functionality

  Rest pages are device-resident initial start pages. They may contain icons, and advertising that is remotely changed by the service provider. Tivoli Personalized Services Manager Device Manager controls the distribution of rest pages to devices.

- APIs

  The device management API allows devices and related information to be maintained in the Device Manager database and be used by external applications, Device Manager servers, and administrative clients.

  The Device management server API allows for configuration and distribution requests to be queued for distribution to devices.

  Device management server API allows the DMS servlet and device plug-ins to communicate.

Table 1 shows the matrix of Device Manager functions and device plug-ins. The rest page function is only supported on NetVista device.

Table 1. DM functions and device plug-ins matrix

	Palm	**Aero 8000**	**NetVista**	**WinCE**
Enrollment	Supported	Supported	Supported	Supported
Device config	Supported	Supported	Supported	Supported
Software distribution	Supported	Supported	Supported	Supported
Rest page support	N/A	N/A	Supported	N/A
APIs	Supported	Supported	Supported	Supported

1.1.6 Supported devices

This section describes the supported devices and device plug-ins that are included in Device Manager 1.1. It also describes device characteristics.

Tivoli Personalized Services Manager Device Manager supports several types of devices such as:

- Personal Digital Assistants (PDAs)
- Screenphones
- Wireless Application Protocol (WAP) devices

Device Manager 1.1 includes an assortment of device plug-ins, which enable Device Manager to manage the following classes of devices:

- Palm Computing PDAs
- Compaq Aero 8000 H/PC Pro devices
- NetVista Internet Appliance devices
- Generic support for Windows CE devices

1.1.6.1 Palm Computing PDAs

Device Manager supports all Palm III and Palm V-series Palm Computing devices that use the PalmOS R3.0.1 (or later) operating system. Palm positions these devices as PC companions used as “connected organizers.” PalmOS devices, with their relatively small displays, infrequent network connections, and portability features, are not necessarily optimized for all the same services that might be offered to users of other devices.

For example, rest page management jobs are not supported on Palm devices due to the limited graphics capability and the device is usually being utilized in disconnected mode.

Device Manager uses the PalmOS APIs and the TCP/IP stack provided by PalmOS 3.0. The Device Manager software includes the Palm device plug-in and the Palm agent program. All PDAs that have the PalmOS 3.0 operating system or higher, use the Palm plug-in and the Palm agent program.

The Palm plug-in installs with the Device Manager software. The plug-in software and the device agent program communicate with each other using a protocol, based on the HTTP or HTTPS protocol and perform system management tasks. You can use SSL by enabling SSL for the Palm agent program.

For more information about the Palm computing devices, see the Palm organizer information on the Palm Computing Web site:

`http://www.palm.com`

1.1.6.2 Compaq Aero 8000 H/PC Pro devices

The Compaq Aero 8000 Handheld PC Professional (Aero 8000) is a handheld PC or sub-notebook for sales and service people, mobile business professionals, and other field personnel who need access to their enterprise network or the Internet. This handheld PC includes Microsoft Windows CE for Handheld PC Professional Edition, Version 3.0 operating system. With the Aero 8000, you can access the Internet or the enterprise network with an Ethernet PC card or internal modem.

The Aero 8000 can be preconfigured for Device Manager by a service provider. The consumer has quick access to the Internet through the Pocket Internet Explorer browser. For more about the Aero 8000, see the Compaq Web site:

```
http://www5.compaq.com/products/handhelds/8000/
```

The Device Manager software includes a device plug-in for the Aero 8000 and its agent program. The Aero 8000 plug-in installs with the Device Manager software. The communication protocol between the Aero 8000 plug-in and its agent program is based on the HTTP or HTTPS protocol. You can use SSL by enabling SSL for the Aero 8000 agent program.

1.1.6.3 NetVista Internet Appliance devices

The NetVista Internet Appliance is a device for Internet access, e-mail, voice mail, Personal Information Management (PIM), and other pervasive computing tasks. This device uses the pervasive computing (PvC) client stack as the base architecture, and then places a shell and applications on the PvC client stack.

The PvC client stack is a set of software components for a wide range of non-traditional devices, such as Service Gateway or Networked Vehicle. The stack consists of a real time operating system, native programs, a JVM, the Service Management Framework, and other services.

To manage the NetVista Internet Appliance efficiently, the plug-in software for Device Manager server and the device agent program are needed. The plug-in software and the device agent program communicate with each other using a protocol, based on the HTTP or HTTPS protocol, and perform system management tasks.

The plug-in software functions as the servlet on the HTTP server and the device agent program is implemented as a Service Management Framework bundle written in Java and functions as the client. When disconnected from the service provider's network, the display on the NetVista Internet Appliance presents a rest page, which is a Web page customized by the service provider to display advertising and other information. The NetVista Internet Appliance can be preconfigured for Device Manager by a service provider.

For more information about the NetVista Internet Appliance, see the NetVista Web site:

```
http://www.pc.ibm.com/us/netvista/index.html
```

1.1.6.4 Generic Windows CE devices

A Windows CE device is a handheld PC, Palm-type device, pocket-type device, or sub-notebook for sales and service people, mobile business professionals, and other field personnel who need access to their enterprise network or the Internet.

This handheld PC includes Microsoft Windows CE for Handheld PC Professional Edition, Version 3.0 operating system. Windows CE devices have resources unavailable with many Palm-type devices. With Windows CE devices, you can access the Internet or the enterprise network with an Ethernet PC card or internal modem.

A Windows CE device can be preconfigured for Device Manager by a service provider. The consumer has quick access to the Internet through the Pocket Internet Explorer browser.

The Device Manager software includes a device plug-in for Windows CE devices and the Windows CE agent program. The Windows CE plug-in installs with the Device Manager software.

The communication protocol between the Windows CE plug-in and the Windows CE agent program is based on the HTTP or HTTPS protocol. The Palm-type WinCE device has no SSL support.

Table 2 shows relationships between device type and supported security functions.

Table 2. Supported security functions

	Palm	**Aero 8000**	**NetVista**	**WinCE**	
				(Palm-type)	**(Other)**
Proxy	Supported	Supported	Supported	Supported	Supported
SSL	Supported	Supported	Supported	N/A	Supported

1.1.7 What's new in DMS 1.1

The enhancements and changes for this release of Device Manager include:

- Job distribution by realm, deal, or both
- Support for DB2 Universal Database Version 7
- Changes to device plug-ins
- Cradle support
- Proxy support
- Enhancements to the Device Manager console

- Support for Japanese language
- Integration with IBM WebSphere Everyplace Suite

1.1.7.1 Job distribution by realm, deal, or both

A device's owner can be the member of a single realm and of one or more deals, as defined by the Subscription Manager component used with Device Manager. It is now possible to distribute jobs to devices in a realm, in a deal, or in both. In addition, an administrator can filter jobs by realm or by deal.

For more information about job distribution, refer to 7.2, “Software distribution” on page 199.

1.1.7.2 Support for DB2 Universal Database Version 7

Device Manager 1.1 provides support for DB2 Universal Database Version 7, in addition to the existing support for Oracle8i Version 8.1.5, as its relational database.

1.1.7.3 Changes to device plug-ins

The following device plug-ins have been added:

- The NetVista Internet Appliance device plug-in provides support for devices used for Internet access, e-mail, voice mail, Personal Information Management (PIM), and other pervasive computing tasks that use the pervasive client stack as their base architecture.
- The generic Windows CE device plug-in provides basic support for all devices that use the Microsoft Windows CE operating system.

1.1.7.4 Cradle support

Device Manager’s built-in Palm device plug-in supports Palm device access using a cradle. To support this connection, Palm plug-in provides the conduit software named TPSM1.1 Palm Cradle Support for Windows. The installation program is shipped with Device Manager.

1.1.7.5 Proxy support

Device Manager 1.1 includes support for communication devices using proxy. Each device can set a proxy server by enabling the ProxyEnable attribute for the device agent program.

1.1.7.6 Enhancements to the Device Manager console

This console's filtering capability allows for more granularity and more complex combinations of filtering criteria. New icons help an administrator to identify which criteria are being used and which are not valid or not in effect.

Other “look and feel” changes improve the usability of the console and its ability to display very large numbers of devices.

For more information on how to use the Device Manager console’s filtering functions, refer to 4.3.2, “Filtering” on page 129.

1.1.7.7 Support for the Japanese language

In addition to single-byte, double-byte character set support is available. Also, the menus, messages, and documentation are translated into double-byte Japanese Kanji. Data and keyboard commands can be entered in Kanji. Double-byte data is stored in the products' databases in double-byte format. The Japanese language is selected when the product is installed.

1.1.7.8 Integration with IBM WebSphere Everyplace Suite

As part of Tivoli Personalized Services Manager Version 1.1, Device Manager supports the WebSphere Everyplace Suite concept of an authentication proxy. The authentication proxy provides a single sign-on to the WebSphere Everyplace Suite servers, including the Tivoli Personalized Services Manager server.

Installers who choose to take advantage of this support need to perform additional steps during Device Manager configuration so that it can redirect devices correctly through the authentication proxy during enrollment.

For more information about installation and configuration in IBM WebSphere Everyplace Suite system, refer to the online manual *WebSphere Everyplace Suite Getting Started v1.1*.

1.2 Why you need DMS

This section establishes the importance of managing pervasive devices.

The next wave of customer demand for e-business will be fueled by Internet access from various pervasive devices. Figure 5 on page 18 illustrates the expected growth in the use of the Internet from these new classes of devices and the number of shipments for each type of device.

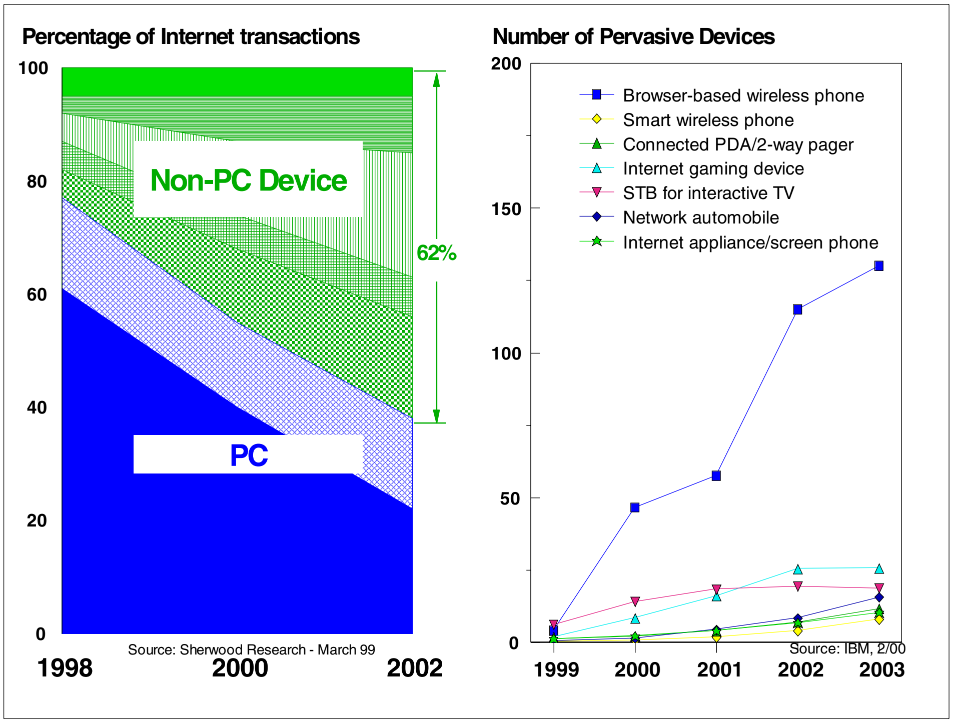


Figure 5. Changes in devices of Internet client

The key feature from Figure 5 is that PC penetration is flattening out because of cost and usability reasons in both the consumer and commercial markets.

To increase the business chance or to provide new business models, service providers must reach out to a larger set of users by making device usage as simple as a wireless phone. They must increase the usefulness of the devices and back-end data by merging personal and business data and providing access from PvC devices, such as the smartphone, PDA, and wireless phone.

There will also be a new class of not-so-personal devices that are more consumer-oriented, such as the in-vehicle information system, home service gateway, kiosks, set-top boxes, and so forth.

However, the functional capability and resources of pervasive devices are normally limited or different between each kind of device. Users sometimes have problems setting up devices, updating software, re-configuring devices, or registering to new services because many of them are novice users. Even if they are not novice users, they may have little knowledge about PvC device details.

From a service provider’s point of view, they need to manage the users' information and devices efficiently to provide better services and increase their profits. Therefore, systems management in the PvC environment is very important and required for both users and service providers.

Device Manager server provides the solution for system management in the PvC world.

1.2.1 The Internet and e-business today

The modern Web environment consists of three-tier systems that are based on a Web browser:

- The Web server receives inbound HTTP requests, manages communication sessions, and routes requests to the appropriate application server or servers.
- The Web application server executes the business logic, manages application sessions, and interacts with the back-end systems.
- The back-end systems include various internal and external services that ground the business processes being implemented by the e-business.

Service providers do not have to consider the client environments because most clients are fat PCs that have similar capabilities and configurations.

To use pervasive devices such as clients, service providers must implement the managing methods of their subscriber’s devices like Device Manager server.

Figure 6 on page 20 shows the image of e-business today.

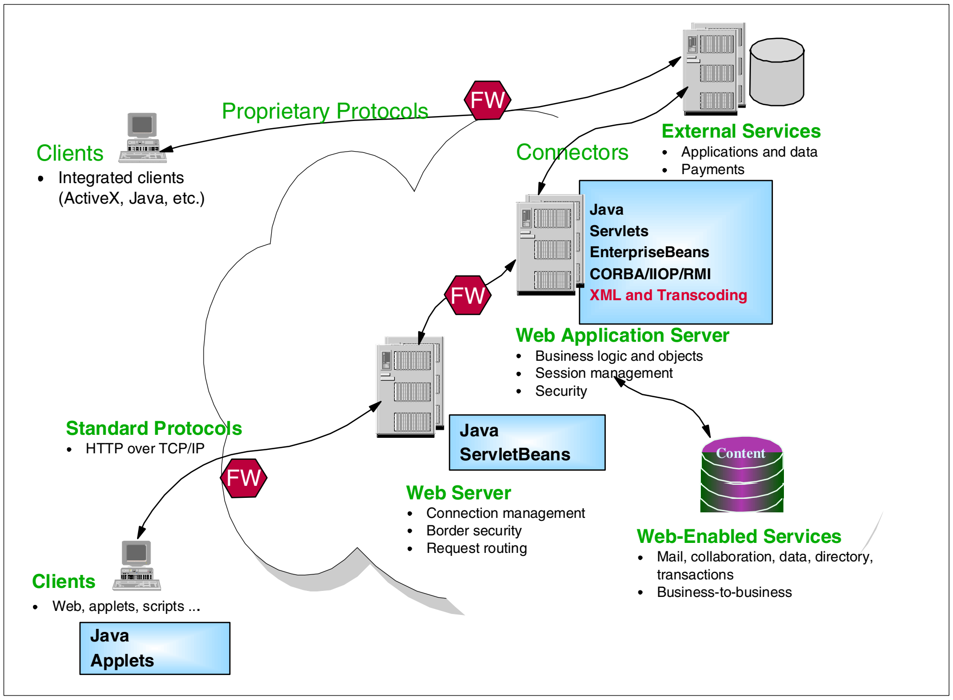


Figure 6. e-business environment today

1.2.2 Pervasive computing and pervasive device management

Pervasive computing is about connecting a variety of non-PC devices to retrieve content, data, and applications from e-business servers.

When compared to e-business solutions today that connect desktop PCs to these servers, pervasive computing introduces the following unique needs:

- Generate and process different types of input/output content.

  Some devices have only text capabilities; others also have graphical capabilities, and others may add or be limited to audio capabilities. The industry is separating content from presentation using XML. Servers must be capable of generating and processing multiple types of markup languages such as subsets of HTML (HTML 3.2, CompactHTML) or other XML presentation languages (WML, VoiceXML).

- Customize content and distribution based on user, device, and network characteristics.

  Content delivered to pervasive devices needs to be personalized, not only to reflect the user's content preferences (content selection and filtering), but also to reflect the environment (reduce content size to match network bandwidth, reformat content to closely match particular target device). Notably, simply selecting the right markup language is insufficient, because different pervasive devices will present that content differently.

- Manage the device from the network to reduce support costs and enhance user experience.

  Pervasive devices are often accessed by users who are not computer-literate, or by mobile workers who are far from the physical control of the enterprise IT department. Consequently, extremely easy-to-use device management is needed to upgrade client software, set up device configuration, and perform other management functions on behalf of inexperienced users.

To integrate pervasive device solutions, service providers must consider the previously mentioned differences in the e-business environment today. There are various types of devices, networks, and user environments to consider.

Figure 7 shows an image of e-business with pervasive devices.

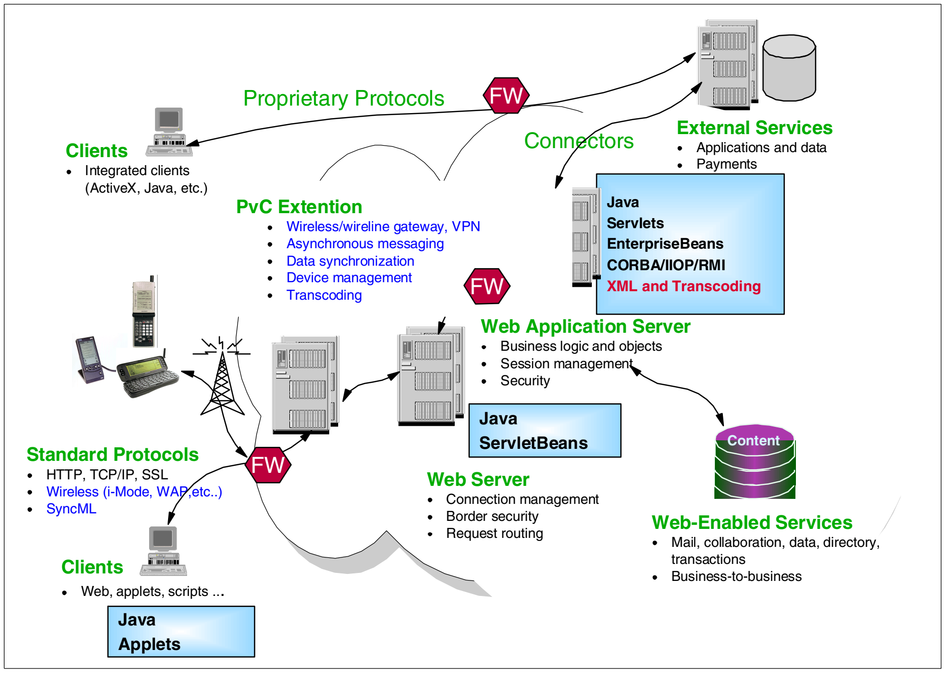


Figure 7. e-business environment with PvC devices

1.2.3 The challenge

To integrate PvC environments into e-business today, service providers have to face two types of management challenges:

- Subscriber's information (managed by Tivoli Subscriber Manager)
- Various types of devices (managed by Device Manager server)

TPSM provides both types of management services. By using TPSM, a service provider can gather and store all information needed to manage their subscribers.

Figure 8 shows the relationship between Tivoli Subscriber Manager and Device Manager server.

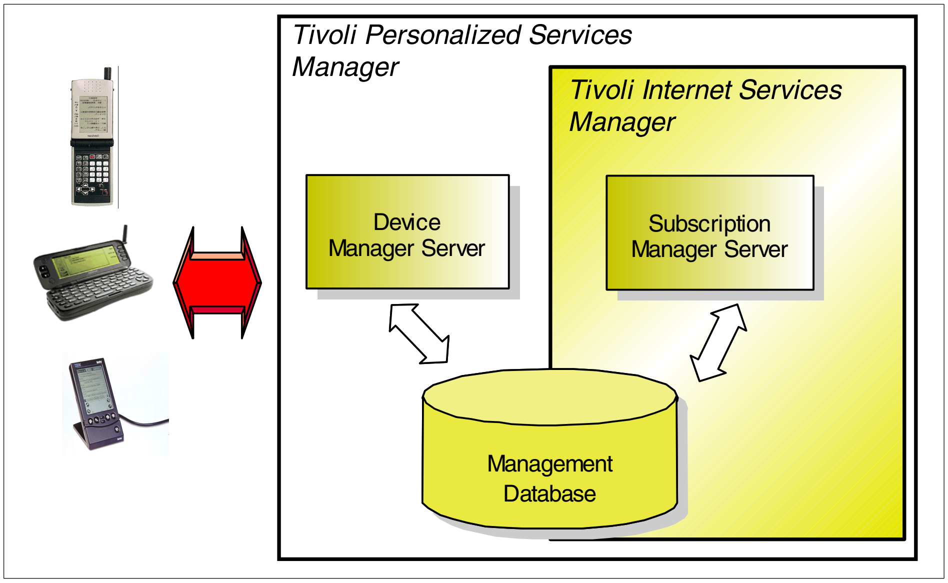

Tivoli Personalized Services Manager
Tivoli Internet Services Manager
Device Manager Server
Subscription Manager Server
Management Database

Figure 8. Tivoli Personalized Services Manager Components

The Tivoli Personalized Services Manager provides a wide range of Internet subscriber management features including:

- Enrollment and self-care
- Personalization
- Authentication and access control
- Customer care and self-care
- Director tool
- Reporting

The Tivoli Personalized Service Manager (TPSM) includes the Device Manager server component and it can extend these services to PvC device environments. DMS provides the following core services:

- Enrolling subscribers and their devices
- Distributing software to the device
- Updating device configuration remotely
- Listing the devices owned by a user
- Updating rest pages (startup pages) for devices

The DMS feature provides a flexible framework and a set of services for managing a subscriber’s devices. Device vendors can extend the system’s framework and use it to enable their devices to be managed by developing their own device plug-ins. TPSM includes the following APIs:

- Device management server API
- Device management API

DMS is software that helps service providers manage their subscribers' information appliances such as Personal Digital Assistants (PDAs), handheld PCs, subnotebooks, smartphones, Wireless Application Protocol (WAP) devices, set-top boxes, in-vehicle information systems, and other, emerging devices for pervasive computing.

Chapter 2. Architecture

This chapter describes how Device Manager is composed, how it works, and how it interacts with other applications such as subscription management.

2.1 Overview

As you can see in Figure 9 on page 26, Device Manager consists of several components:

- Device Manager database
- Device management server servlet
- Device plug-ins and agents
- Device Manager console
- Device management API
- Device management server API

Throughout this redbook, we may refer to Device Manager components and other related components as follows:

- **Device Manager database**: DM DB
- **Device management server servlet**: DMS servlet
- **Device plug-in**: plug-in
- **Device agent**: agent
- **Device Manager Console**: DM console
- **Subscription Manager**: SM

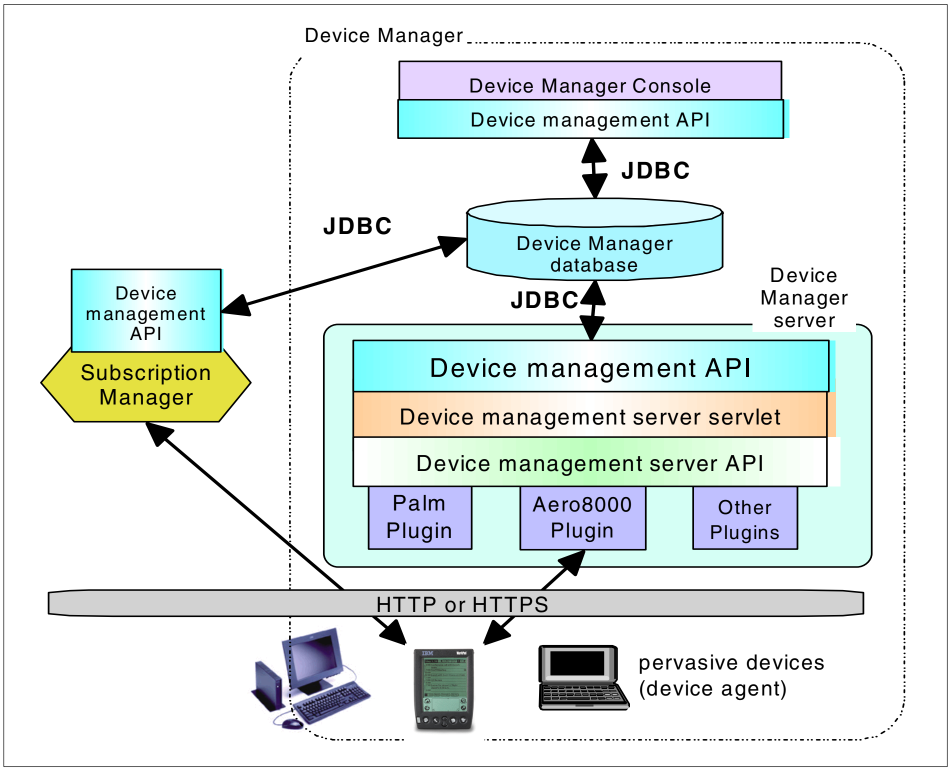


Figure 9. Device Manager architecture

Device Manager has a Device Manager database as its core component. The Device Manager database stores all device management information as follows which is used to process device management jobs:

- Device related information
  - Devices
  - Device classes
  - Device parameters
- Software information
  - URL of the software package to be distributed
  - Version of the software
- Job related information
  - Job classes
  - Submitted jobs

Device Manager server, in the middle of Figure 9, processes the device management tasks by using the data in the Device Manager database. Device Manager identifies the devices connected and then redirects them to the enrollment application, for example: enrollment service in TPSM Subscription Manager, if it is not yet enrolled, or to the device management server servlet if already enrolled. Device management server servlet actually performs the base management tasks requested by device plug-ins.

Device Manager needs external device enrollment application. In TPSM, it is the enrollment application server which is a feature of Subscription Manager. The Subscription Manager's enrollment application server accesses Device Manager database by using device management API (DM API).

Device Manager can support various types of devices by using device type specific plug-ins. It is device plug-ins that actually perform device-specific management functions. For example, Device Manager's built-in device plug-ins provide management functions such as software distribution, device configuration, and rest page management. Device plug-ins communicate with the DMS servlet using the device management server API (DMS API). DMS API is open to the public, so that device vendors or system integrators can develop their own device plug-ins using the DMS API.

The device management server servlet is designed to coordinate the processing of scheduled jobs on devices as they connect to the network. Currently, the supplied job types are software distribution, device configuration, and rest page management. Each of these job types is implemented in a job class, and these job classes are somewhat device class specific. The Device Manager database maintains the table of job definitions and the device management API allows new job types and their associated job classes to be defined in this job table. This allows enterprises, ISPs, integrators, or device manufacturers to provide custom job classes for new job types, to allow value-added product differentiation.

A device agent to be installed on the device knows how to communicate with the associated Device Manager plug-in. This is developed together with each device plug-in.

Business administrators and system administrators use Device Manager console to perform management tasks. They can submit jobs, control device parameter settings, and manage software to be distributed. This console accesses and updates the Device Manager database through the DM API.

To execute a job for one or more devices, a job is submitted for the device or devices using the Device Manager console or using the job management API.

At the time the job is submitted, the administrator or application specifies information such as the job type to be processed, job-specific parameters, target devices, and the activation and expiration time of the job. This job-related information is stored in the Device Manager database. For more information on the Device Manager database, refer to Chapter 5, “Device Manager database” on page 145. For more information on submitting jobs, refer to Chapter 4, “Administration” on page 109.

2.1.1 Features of Device Manager

This section describes the features of Device Manager.

2.1.1.1 Extensibility for managing various devices

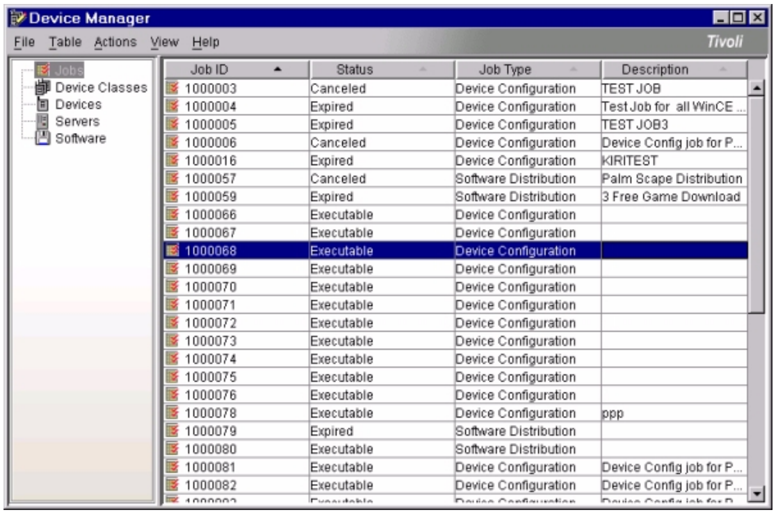
To support extensibility to manage a variety of devices, the Device Manager server supports a “plug-in” architecture. Device Manager provides some basic device modules that can be easily extended to support various device types, and it can support more than one device type at the same time.

TPSM Device Manager provides device plug-ins that support Palm Pilot, Aero 8000, general Windows CE, and NetVista Internet Appliance by default. Other device vendors, system integrators or ISPs can develop their own plug-ins that have specialized functions to meet their needs. Device agents are provided with each device plug-in. A device agent resides on the device, and communicates with the server side plug-in. The communication between a device plug-in and the device agent is device-class-dependent and is determined by the integrator that writes the plug-in code. For example, a WAP device vendor can develop a “WAP device plug-in” and the agent that uses WAP as its communication protocol between the plug-in and the agent. TPSM Device Manager plug-ins, which support Palm Pilot, Aero8000, general Windows CE devices, and NetVista Internet Appliance, use HTTP and HTTPS as their transport mechanism between the plug-in and the device agent. This allows device management traffic to pass through various kinds of network elements such as firewalls. Interactions take the form of request/response pairs initiated by the device.

The device management server API defines the programming interface between a device plug-in and the device management server servlet. The device management server API is designed to be generic and protocol-independent to support many kinds of devices and jobs.

2.1.1.2 Centralized management

Device Manager console provides a graphical interface to manage devices, software and jobs as shown in Figure 10.



Job ID	Status	Job Type	Description
1000003	Canceled	Device Configuration	TEST JOB
1000004	Expired	Device Configuration	Test Job for all WinCE ...
1000005	Expired	Device Configuration	TEST JOB3
1000006	Canceled	Device Configuration	Device Config job for P...
1000016	Expired	Device Configuration	KIRITEST
1000057	Canceled	Software Distribution	Palm Scape Distribution
1000059	Expired	Software Distribution	3 Free Game Download
1000066	Executable	Device Configuration	
1000067	Executable	Device Configuration	
1000068	Executable	Device Configuration	
1000069	Executable	Device Configuration	
1000070	Executable	Device Configuration	
1000071	Executable	Device Configuration	
1000072	Executable	Device Configuration	
1000073	Executable	Device Configuration	
1000074	Executable	Device Configuration	
1000075	Executable	Device Configuration	
1000076	Executable	Device Configuration	
1000078	Executable	Device Configuration	ppp
1000079	Expired	Software Distribution	
1000080	Executable	Software Distribution	
1000081	Executable	Device Configuration	Device Config job for P...
1000082	Executable	Device Configuration	Device Config job for P...

Figure 10. Device Manager console

Business administrators and system administrators use only this console to perform management tasks. They can perform almost all management tasks from this tool. In addition, Device Manager console can cooperate with Authentication Server of Subscription Manager and provide authentication functions so that only valid administrators can use it.

2.1.1.3 Provisioning to other systems

Device management API provides the programming interface for managing devices, jobs, and related resources in the Device Manager database. All of the Device Manager components and all the other applications that access the Device Manager database should use this API. For information about device management API, refer to Chapter 6, “Application programming interface” on page 171.

The interfaces between the Device Manager and the Subscription Manager are well defined and highly localized. Consequently, DM can be used as a facility through which other applications can perform device management. Essentially, the DM system is a repository of device data and an engine for processing jobs on devices when they are connected to the network. DM relieves the applications that use it from the need to store detailed device information such as operating system, installed software, and configuration settings, and shields them from the protocols used to communicate with each type of device.

2.1.1.4 Scalability and availability

Scalability is achieved horizontally, with many DM servers identically configured. Requests are routed to these servers by a front end such as IBM Network Dispatcher.

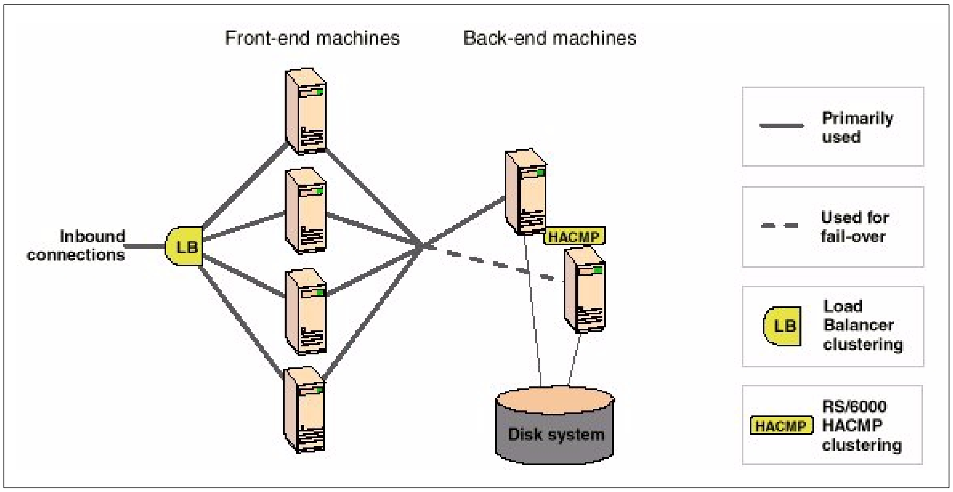


Figure 11. Scalability and availability

Network Dispatcher also allows adding some extra Device Manager servers, without stopping the service.

For the back-end database system, you can use the IBM AIX HACMP solution for high availability.

2.2 Components of Device Manager

This section describes Device Manager components in detail. They are:

- Device management server servlet
- Device Manager database
- Device Manager console
- Device plug-ins
- Device management API
- Device management server API

2.2.1 Device management server servlet

When a device connects to the service provider's network, the device management server servlet (DMS servlet) ensures that the device is enrolled.

If it is, it coordinates the processing of all scheduled device management jobs for the device. Jobs are job classes included with each device plug-in that resides on the Device Manager server. Device Manager supports job types, such as device configuration, software distribution, and rest page management, which interface with the DMS servlet using the device management server API.

Jobs are submitted for processing by an administrator using the Device Manager console or by another application using the device management API. At the time the job is submitted, the administrator or application specifies the type of job, any job-specific parameters, the devices the job should run on, the activation and expiration time for the job, and other information required by the job.

When a device connects to the service provider's network, it communicates either directly or through a Network Dispatcher to a Device Manager server for job processing. Using device management server API calls, the DMS servlet checks to make sure the device is enrolled with the service provider and redirects it as needed.

- For a device requiring enrollment, the function redirects the device to the service provider's enrollment server for registration.
- For an enrolled device, if there are jobs pending for it to run, the function redirects the device back to the Device Manager server. This prevents the device from returning through a Network Dispatcher during job processing and then being rerouted to a different Device Manager server, should multiple HTTP connections be required to complete the job.

The DMS servlet searches the database, again using the device management API, for any submitted jobs eligible to be run that pertain to the device and builds a prioritized job list. To run the jobs, the DMS servlet uses device management server API calls through the device plug-in to interact with the device, often through several iterations of requests and responses.

2.2.2 Device Manager database

Device Manager database stores all device management information which is used to perform management tasks. When a device connects to the Device Manager server for the first time, the device information is enrolled to the Device Manager database, such as device ID, device type and user name. If software distribution jobs are submitted to some devices, the information about the job, software package name, distribute duration, target realm, and so forth are stored.

Device Manager database is accessed using the device management API. This is the way to correctly update the records in the Device Manager database related to each other. Therefore, device management server servlets, Device Manager console or all external applications such as Subscription Manager and billing system should use device management API to access Device Manager database. For detailed information about the Device Manager database, see Chapter 5, “Device Manager database” on page 145.

2.2.3 Device plug-ins

A device plug-in resides on the Device Manager server and provides the logic that handles device identification, communications, job processing, and high-level management tasks for a particular class of managed devices. Put simply, a plug-in defines support for a Device Manager device class. A device class defines a group of devices whose operations can be managed similarly. For example, one of the device classes included with Device Manager defines Palm PDAs that run Palm OS 3.0.1 or later. Device plug-ins are typically provided with Device Manager, but may also originate with a device manufacturer or integrator and be installed later on a Device Manager server.

The device plug-ins communicate with device agents (device-specific software that is installed on the device), distribute software, perform remote device configuration, or perform other types of jobs. The communications between a device plug-in and the device agent is device-class-dependent and is determined by the integrator who writes the plug-in code. The device management server API defines the programming interface between a device plug-in residing on Device Manager and the rest of Device Manager, including its redirection function. The device management server API is designed to be generic and protocol-independent to support many different kinds of devices and jobs.

2.2.3.1 The components of device plug-in

Plug-ins are logically structured as four pieces:

- A device-specific communication component that the plug-in software uses to communicate with the device. This interface is defined by the plug-in developer.
- The internal device communications component, which enables communication between the plug-in software and DMS servlet. This includes a DeviceCommunicationManager implementation.
- The set of device-specific job classes. This piece of the plug-in interacts with the DMS servlet, the device-specific communication component, and

the DeviceCommunicationManager implementation. It implements the higher-level management tasks (such as software distribution or device configuration). This piece is created by the plug-in developer, by implementing the DeviceJob interface.

- An optional template file that describes the set of device parameters such as the IP gateway or domain name server used, that can be configured for all devices of the same device class.

Both DeviceCommunicationManager and DeviceJob are abstract Java classes in the device management server API that the plug-in writer extends as part of developing device classes and device jobs.

Device plug-ins provide the device-specific implementation logic for devices managed by the Device Manager. Device plug-ins implement the logic described in the first two bullet points in the previous list as the device class layer. It defines a set of methods to perform low-level primitive operations on the device. The job class layer implements high-level management tasks, which is described in the third bullet in the previous list. It performs management tasks by invoking the methods of the device class layer.

Device class

Each type of device managed by the DM has its own device class implementation, and the DM assumes that all devices managed by a particular device class are capable of running the same set of software applications. In other words, when a software entry is defined in the device management database, it is associated with a single device class, and all devices managed by that device class are assumed to be able to run that software. If an application can run on devices in more than one device class, a software entry must be defined for each device class.

For device classes that use HTTPS as the protocol, the Web server infrastructure can provide encryption of sensitive data such as userID and password, but device classes that use some other protocols are responsible for ensuring that the data is transmitted to devices in a secure manner. No such security mechanism will be provided at the DM level.

Each device class may provide a template file that describes the device-specific configuration parameters required by the devices it manages. This template file will define the list of parameters, their syntax, and information to assist the DM console in building a GUI to present the parameters to an administrator. This GUI will allow the parameters to be viewed or modified for a specific device or for the device class.

Job class

Job classes implement the logic to perform specific high-level management tasks. For the initial release, job classes will be provided to perform software distribution, device configuration and rest page management. The job classes perform these high-level tasks by interacting with the other elements of the device plug-in. For this reason, job classes are likely to be device-class specific.

The instance of job class is created by the DMS servlet in order to process jobs for a specific device.

The job class can get the job parameters, the device parameters, the submitted job ID, and other information which is previously set by the DMS servlet's initJob call.

The job class can set or get the job context object so that any specific information can be maintained as different portions of the job are executed. The format of the job context can be freely defined inside the job class.

The job class implements the "doJob" method to process the job by invoking the APIs to the device class. The doJob method is called by the DMS servlet. This method's parameter is the device context object passed through the DMS servlet. The device context object indicates the device and device-specific state or status information. The format of the device context can be defined between the job class and the device class.

Template

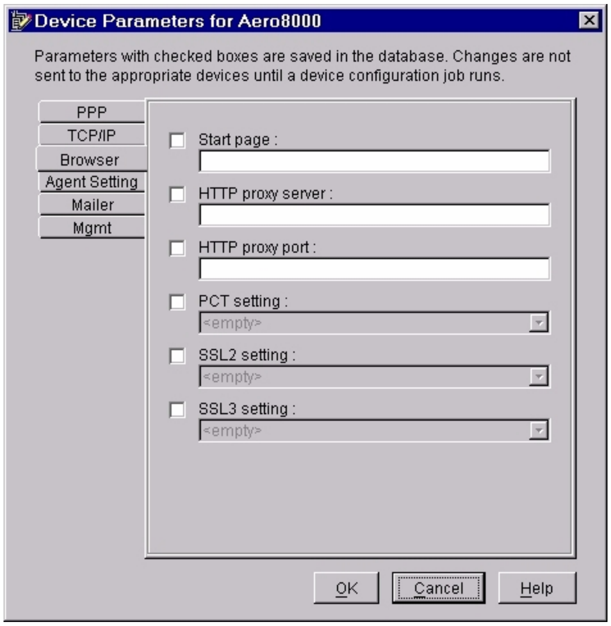
The template file defines the set of device-specific parameters required for a device. The template file is optional, so if a plug-in writer is creating a plug-in for a device class that does not have any configurable parameters, a template file is not required.

When the device class is registered to the DM server, the template file for the device class is parsed and the parameter definitions are stored in the DEV_CLASS_TEMPLATE table of the DM DB. If the file is not provided, device configuration in the DM console for devices of this class will not be supported.

The DM console uses the device management API to retrieve the parameter definitions for a particular device class and build a device configuration dialogue as shown in Figure 12. The dialogue consists of at least one tab. Each tab includes device parameters such as text fields or combo boxes. Configuration data (values) are stored in the DEVICE_PARMS and

DEVICE_CLASS_PARMS tables and passed to a job class when a job is processed for a device.

The format of the template file is described in Chapter 7, “Using the DM functions” on page 193. Appendix A, “Device parameters” on page 233, has the list of the parameters provided by the built-in plug-ins. The information on how you can use the device class configuration utility is described in Chapter 3, “Installation and configuration” on page 55.



Device Parameters for Aero8000

Parameters with checked boxes are saved in the database. Changes are not sent to the appropriate devices until a device configuration job runs.

□ Start page :

□ HTTP proxy server :

□ HTTP proxy port :

□ PCT setting : <empty>

□ SSL2 setting : <empty>

□ SSL3 setting : <empty>

Figure 12. The parameter setting dialogue for Aero8000 device class

Device agents

Device agents are also provided by plug-in vendors. This is the software installed on the device to communicate with the server side device plug-in to perform work.

Device agents can be written in any programming language. This should be decided in consideration of performance, productivity, or other possible conditions.

2.2.4 Device Manager console

The Device Manager console is a graphical user interface (GUI) to administer device management operations from a Microsoft Windows client as shown in Figure 13 on page 36. Administrators can use this interface to perform tasks like add and view devices and device software, submit jobs for devices, and query job status. The console invokes methods of the device management API to access information in the Device Manager database and perform requested operations.

From the DM console, a service provider's administrators can manage:

- Single devices
- Classes of devices
- Parameters for devices
- Parameters for device classes
- Jobs for specific devices
- Jobs for all devices in a particular realm
- Jobs for all devices in a particular deal
- Software for devices
- Device Manager servers

Typical jobs for devices include the following supplied job types though all job types may not apply to all devices:

- Device configuration
- Software distribution
- Rest page management

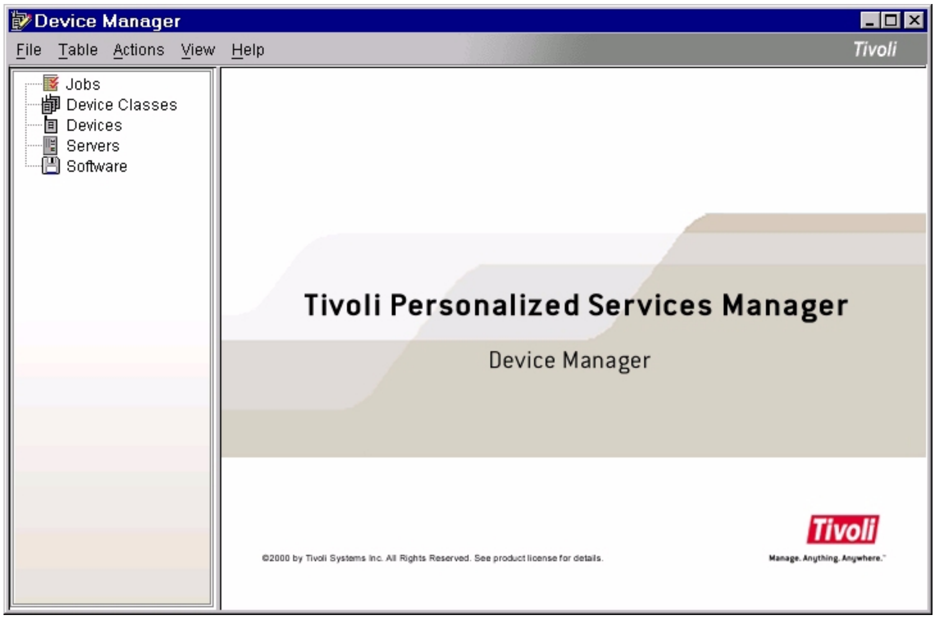

Device Manager
File
Table
Actions
View
Help
Tivoli
Jobs
Device Classes
Devices
Servers
Software
Tivoli Personalized Services Manager
Device Manager
©2000 by Tivoli Systems Inc. All Rights Reserved. See product license for details.
Tivoli
Manage. Anything. Anywhere.

Figure 13. Device Manager Console

Administrators submit these jobs to update the configuration of devices including network parameters, distribute new or updated software applications to devices, and update device rest pages with, for example, timely new information from the service provider.

Administrators can initiate a new job of an available job type and target it to:

- All devices
- All devices of a device class
- All devices in a realm, a deal, or both
- One or more selected devices
- Other typical administrator tasks including:
  - Viewing device jobs and their status, and canceling jobs
  - Identifying and configuring a newly deployed device, including associating the device with its owner
  - Obtaining and displaying information about the software Device Manager has distributed to a device, and the configuration parameters it has set for a device

More information about Device Manager console and its uses may be found in Chapter 4, “Administration” on page 109.

2.2.5 Device management API

Device management API provides the programming interface for managing devices, jobs, and related resources in the Device Manager database. The device management APIs will be used by the other Device Manager components, such as DMS servlet and DM console. In addition, the device management APIs will be used by various pieces of the Subscription Manager (SM) component of TPSM. Examples of how the device management APIs are (or can be) used by these components are:

- DM console will use DM API to perform the various operations that are invoked by an administrator, such as creating/listing devices, displaying device classes, viewing/modifying the parameters for devices or device classes, and submitting new jobs for devices.
- SM component of TPSM will use DM API to define, configure, and schedule jobs for new devices enrolled through the enrollment application. In addition, customer care or self-care applications can be written to enable customer support representatives (CSRs) or customers to view or manage devices within their scope of control.

For more detailed information on device management API, refer to Chapter 6, “Application programming interface” on page 171.

2.2.6 Device management server API

Device management server API (DMS API) defines the programming interface between device management server servlet (DMS servlet) and plug-ins. This interface allows plug-ins to interact with a DMS servlet. DMS API is a general purpose, protocol-independent interface that serves as an “abstraction layer”, allowing all classes of devices, though they may operate differently, to be managed in the same way by Device Manager. Either Device Manager, or the device manufacturer or integrator, provides a device plug-in for the new device that implements the required functionality and contains logic for performing the unique management functions for the class of devices. DMS API can be used to develop plug-ins for any new class of plug-ins to implement device identification, communications, and job processing for all devices in the new device class.

For more detailed information on device management server API, refer to Chapter 6, “Application programming interface” on page 171.

2.3 Control flow

This section describes how Device Manager controls the device connection and job processing.

There are many package names, class names, interface names, and other specific names in the following description. Table 3 shows the list of them and their description by category.

Table 3. The list of names

Name		Description
General		
1	device ID	The identifier of the device
2	realm	It is a partition within the overall database that distinguishes them and their subscribers from all others in the system and the Internet at large. Realms help extend brand name identification.
DMS servlet		
3	fire(DeviceJobProcessing CompleteEvent)	A method in class DeviceCommunicationManager (DCM). It is called by a DCM to notify the DMS servlet that it has completed a DeviceJob.

Name		Description
4	fire(DeviceRequestWork Event)	A method in class DCM. It is called by a DCM to notify the DMS servlet that a device is ready to perform run or continue running a DeviceJob.
plug-in: device class		
5	Device-specific communication software	Typically, it represents the class which extends DeviceCommunicationManager so that it can manage the communication for the specific device type.
6	device class	Generally, device class means the group of devices which have similar characteristics.
7	DeviceCommunication servlet	The same meaning as Device-specific communication software.
8	DeviceCommunication Manager	This class describes the functionality to be extended by a plug-in developer or device manufacturer to enable a class of devices to be managed by Device Manager. This class provides a common interface for the Device Manager to communicate with plug-ins supporting different types of devices.
9	DeviceConnectionEvent event	It is typically called by a DCM to notify the DMS servlet that a device has established or closed a connection.
10	device class API	The APIs provided by device classes and used by job classes.
11	device context object	It represents an actual instance of a device. It is created and managed by associated device class. It mainly works as device specific context holder during job session.
12	getPackageData	An API provided by the built-in plug-in device classes. The software distribution job class invokes it to obtain the software package data.
13	putPackage	An API provided by the built-in plug-in device classes. The software distribution job class invokes it to put the software package to the device.
14	redirectToDevice ManagementServer	A method in the DeviceCommunicationManager class. Redirect the device to the DM server on the specified host and port.

Name		Description
15	redirectToEnrollment Server	A method in the DeviceCommunicationManager class. It is called by the DCM to instruct the DCM to redirect the last client request to the indicated subscriber manager enrollment server.
plug-in: job class		
16	DeviceJob	This class describes the basic functionality to be extended so that device specific jobs can be created and managed by the Device Manager.
17	doJob()	Method in class DeviceJob. It is called by the job management software so that the device specific portion of the job (or the next part of the job) may be executed.
18	initJob	Method in class DeviceJob. It is called by the job management software after the DeviceJob is created and before it is passed to the DMS servlet for execution.
19	job class	Generally, we call the classes that override the DeviceJob class as job class.
20	job context object	The job context includes the job class state and information that are saved at the previous process.
21	performPostProcessing() method	A method in class DeviceJob. DMS servlet calls this method to perform tasks required after complete the job.
22	performPreProcessing() method	A method in class DeviceJob. DMS servlet calls this method to perform tasks required before start the job.
23	software distribution job class	A kind of job class, which extends the DeviceJob class.

2.3.1 Connection handling

Device Manager handles connected devices according to the device status.

2.3.1.1 New device connection

Devices new to Device Manager should be enrolled to the Device Manager database at the first connection to Device Manager.

Assumption

In this case, the device is assumed to satisfy the following conditions:

- A Palm device is used.
- The device user has already registered to an ISP by telephone or Web page. That is, the user is enrolled to the TPSM database as a subscriber in advance of the first connection to Device Manager.

Connection flow

At first, the user invokes the device agent by clicking the agent icon. Figure 14 shows the steps in a connection flow.

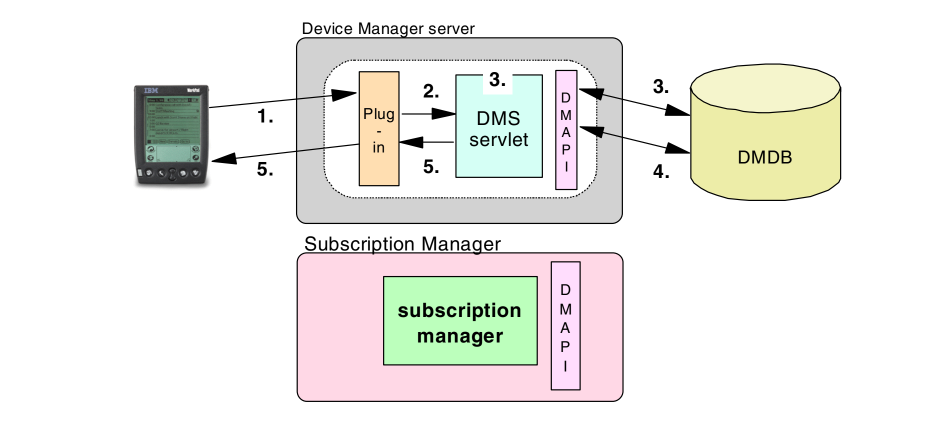


Figure 14. DMD redirects the agent to Subscription Manager

The flow in Figure 14 is explained here:

1. The device agent connects to the Device Manager server. The device class interacts with the device agent to determine a unique device identifier (device ID). When the Device Manager has built-in plug-ins, the HTTP headers contain sufficient information to derive this device ID.
2. The plug-in sends a CONNECT event to the DMS servlet.
3. The DMS servlet checks the cache of the device ID. If the ID of this device is not found, it checks DM DB to see if the device has already been The
4. The DMS servlet gets the enrollment URL from the DM DB, and calls a redirectToEnrollmentServer method on the plug-in to request it to redirect the request. Together with this, the DMS servlet passes the URL of the DM server so that any Network Dispatcher is bypassed and the targeted DM server is accessed directly.
5. The plug-in sends a redirection message with the enrollment URL.

Figure 15 shows how the Subscription Manager enrolls the device to the DM DB.

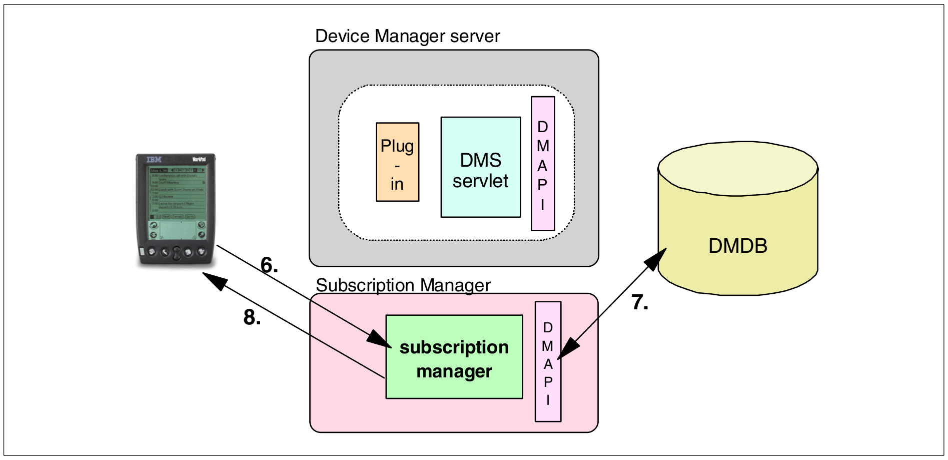


Figure 15. Device enrollment to the Subscription Manager

The process flow in Figure 15 is explained here:

6. The agent is redirected to the Subscription Manager with device-specific information such as device ID, user name, and realm.
7. The Subscription Manager associates the device to an already enrolled user, and creates a device entry for it in the DM DB through the DM API. Then, it submits a device configuration job for the device as an initial configuration job.
8. The Subscription Manager sends the command to the device to redirect to the DMS servlet.

The TPSM Subscription Manager feature implements these steps to cooperate with Device Manager. In general, DM requires that this enrollment application perform the following general steps:

1. Obtain the device ID and device class information from the URL parameters or a cookie passed with the request.
2. Create the device entry for the device ID in the DM DB with the DM API using the information such as device ID, device class, and the subscriber.
3. Submit a job for the new device that will perform the initial device configuration. Any subscriber specific information needed by this job, such as the user ID and password that the device should use when connecting

to the network, should be provided as job parameters when this job is submitted.

4. Redirect the device back to the DMS servlet.

The reason why SM performs device enrollment

Device Manager will not create a device entry in the device management database because the device has not yet been assigned to a subscriber.

2.3.1.2 Enrolled device connection

Devices already enrolled have their entry in the DM DB. When devices already enrolled to the DM DB connect to the Device Manager server, they are redirected to the DMS servlet, bypassing any Network Dispatcher to perform work.

Assumption

The following is assumed for an enrolled device connection:

- A user uses a Palm device.
- The device is already enrolled to the DM DB.

Connection flow

Figure 16 shows the flow for an enrolled device connection.

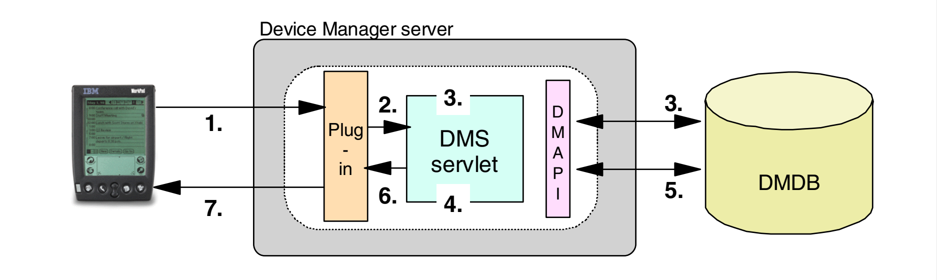


Figure 16. An enrolled device connection

The flow in Figure 16 is explained here:

1. The agent connects to the Device Manager server. The device class interacts with the device agent to determine the device ID. When the Device Manager has built-in plug-ins, the HTTP headers contain sufficient information to derive this device ID.
2. The plug-in sends a CONNECT event to the DMS servlet.

3. The DMS servlet checks the cache of the device ID. If the ID of this device is not found, it checks the DM DB to see if the device has already been enrolled to the DM DB. In this case, it is already enrolled.
4. The ID of this device is added to the device ID cache.
5. The DMS servlet checks to see if there is any job for the device. In this case, there is a job submitted for this device.
6. The DMS servlet calls the plug-in to redirect the device directly to the DM server.
7. The plug-in redirects the device to the DM server.

2.3.1.3 Enrolled device connection without submitted job

If there are no submitted jobs for the connected device, the DMS servlet handles all the job sequences. The device is never redirected to the DMS servlet to bypass the Network Dispatcher.

Assumption

The following points are assumed for enrolled device connections without submitted jobs:

- A user uses a Palm device.
- The device is already enrolled to the Device Manager database.
- This is the fifth connection to the Device Manager server.
- No jobs are submitted for this device.

Connection flow

Figure 17 shows the connection flow for enrolled device connections without submitted jobs.

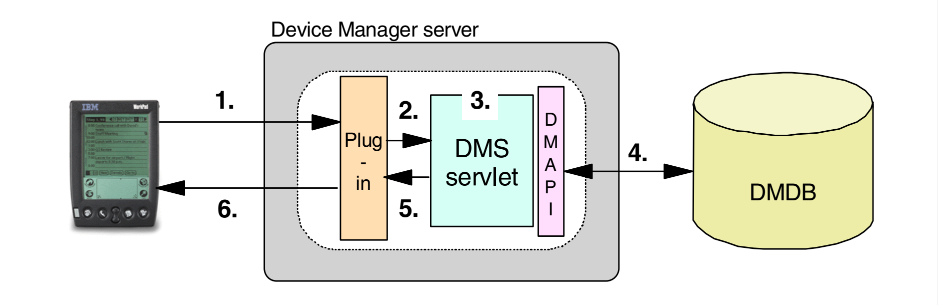


Figure 17. No jobs are submitted for this device

The process flow in Figure 17 is explained here:

1. The agent connects to the Device Manager server. The device class interacts with the device agent to determine the device ID. When the Device Manager has built-in plug-ins, the HTTP headers contain sufficient information to derive this device ID.
2. The plug-in sends a CONNECT event to the DMS servlet.
3. The DMS servlet checks the cache of the device ID. This time, the ID of this device is hit in the cache.
4. The DMS servlet calls a DM API and checks to see if there are any jobs for the device. In this case, there are no jobs for this device.
5. The DMS servlet calls the plug-in to tell the device that no jobs are submitted.
6. The plug-in tells the device that no jobs are submitted, and the connection ends.

Front-ended Network Dispatcher

Once a device connects to the Device Manager server, Network Dispatcher is bypassed so that only one Device Manager server is used in the connection.

In cases where the connected device is new to the Device Manager server, the DMS servlet redirects the device to the enrollment server.

In cases where the connecting device is already enrolled, the DMS servlet redirects it to the DM server directly, bypassing the Network Dispatcher.

2.3.2 Job flow

Device Manager's built-in plug-ins have three functions:

- Software distribution
- Device configuration
- Rest page management

This section shows the general control flow of the job.

Figure 18 on page 46 shows a typical flow of events when a device connects to Device Manager. The device is already enrolled, and is not redirected to an enrollment server. A Network Dispatcher is in place to provide load-balancing and fault tolerance.

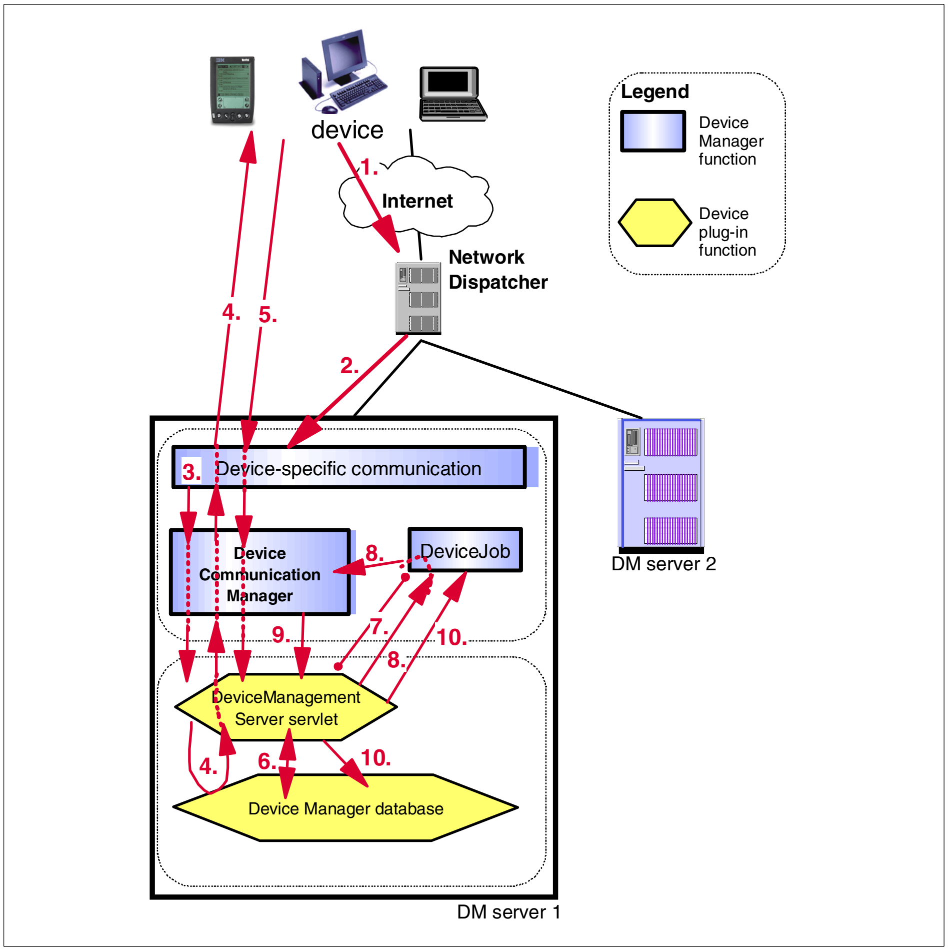


Figure 18. Typical Device Manager job flow

The job flow in Figure 18 is explained here:

1. A device sends a request to connect. The request comes in through the cluster URL of the Network Dispatcher.
2. The Network Dispatcher routes the device request to an available Device Manager in its cluster. Device-specific communication software, as defined by the plug-in, receives the communication from the device.
3. Device-specific communication software passes information to the plug-in's DeviceCommunicationManager implementation. This DeviceCommunicationManager implementation creates a DeviceConnectionEvent event and calls the base

DeviceCommunicationManager method to fire the event, which sends it to the core Device Manager functionality.

4. Since the device is already enrolled, the DMS servlet checks the DM DB to see if there is a job for the device to run. Assuming that it finds one or more jobs ready to be run by the device, the DMS servlet calls the redirectToDeviceManagementServer method to reroute the device directly to Device Manager, bypassing the Network Dispatcher.
5. The device-specific communication for the device receives a message from the device (in this example, a message requesting a job). The device-specific communication calls the DeviceCommunicationManager. The DeviceCommunicationManager creates the appropriate event and calls the fire(DeviceRequestWorkEvent) method implemented in the base DeviceCommunicationManager class to send the event to the DMS servlet.
6. The DMS servlet checks the DM DB to see if there is a job for the device to run.
7. When the DMS servlet finds a job, it creates and initializes an instance of the deviceJob object and calls the performPreProcessing() method to check for any required job and device parameters.
8. The DMS servlet calls the doJob() method on the DeviceJob object. The DeviceJob, along with the other plug-in components and the device, perform the requested job.
9. The DeviceCommunicationManager calls the fire(DeviceJobProcessingCompleteEvent) method when the job is completed.
10. The DMS servlet takes DeviceJob and calls the performPostProcessing() method for it, and updates the DM DB with current device and job status, such as job completed.

2.3.3 Sample scenario

This section describes a sample scenario, from connecting to the server to finishing the job execution.

2.3.3.1 Assumptions

This scenario assumes a Palm device connection. The device has already been enrolled to the Device Manager. A software distribution job has been submitted for this device.

2.3.3.2 Connection and job processing

This flow assumes the Palm device case, but the control flow is common to all other device classes which are built in to Device Manager. Before the device connects to the DM server, administrators perform some management tasks which are illustrated as arrows a, b, and c in Figure 20:

- **Arrow a**: The administrator creates a software package and places it on the file server.
- **Arrow b**: The administrator registers the software to the Device Manager database from DM console.
- **Arrow c**: The administrator submits a software distribution job for the device class from DM console.

When the agent connects to the DM server, the DM server handles the connection as illustrated in Figure 19.

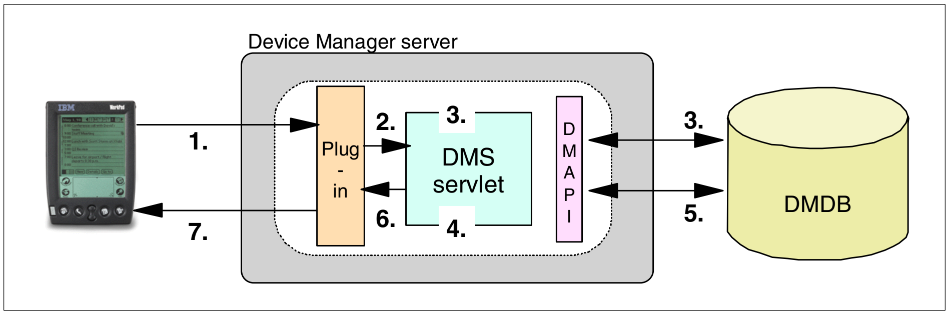


Figure 19. Connection flow

The numbered arrows in Figure 19 are explained here:

1. The agent connects to the Device Manager server. The device class interacts with the device agent to determine the device ID. In the Device Manager built-in plug-ins case, the HTTP headers contain sufficient information to derive this device ID.
2. The plug-in sends a CONNECT event to the DMS servlet.
3. The DMS servlet checks the cache of the device ID. If the ID of this device is not found, it checks the DM DB to see if the device has already been enrolled to the DM DB. In this case, it is already enrolled.
4. The ID of this device is added to the device ID cache.
5. The DMS servlet checks to see if there are any jobs for the device. In this case, there is a job submitted for this device.

6. The DMS servlet calls the plug-in to redirect the device directly to DMS.
7. The plug-in redirects the device to DMS.

Then, the agent connects directly to the DM server again. Since a software distribution job is submitted for this device, DM server processes the job as illustrated in Figure 20.

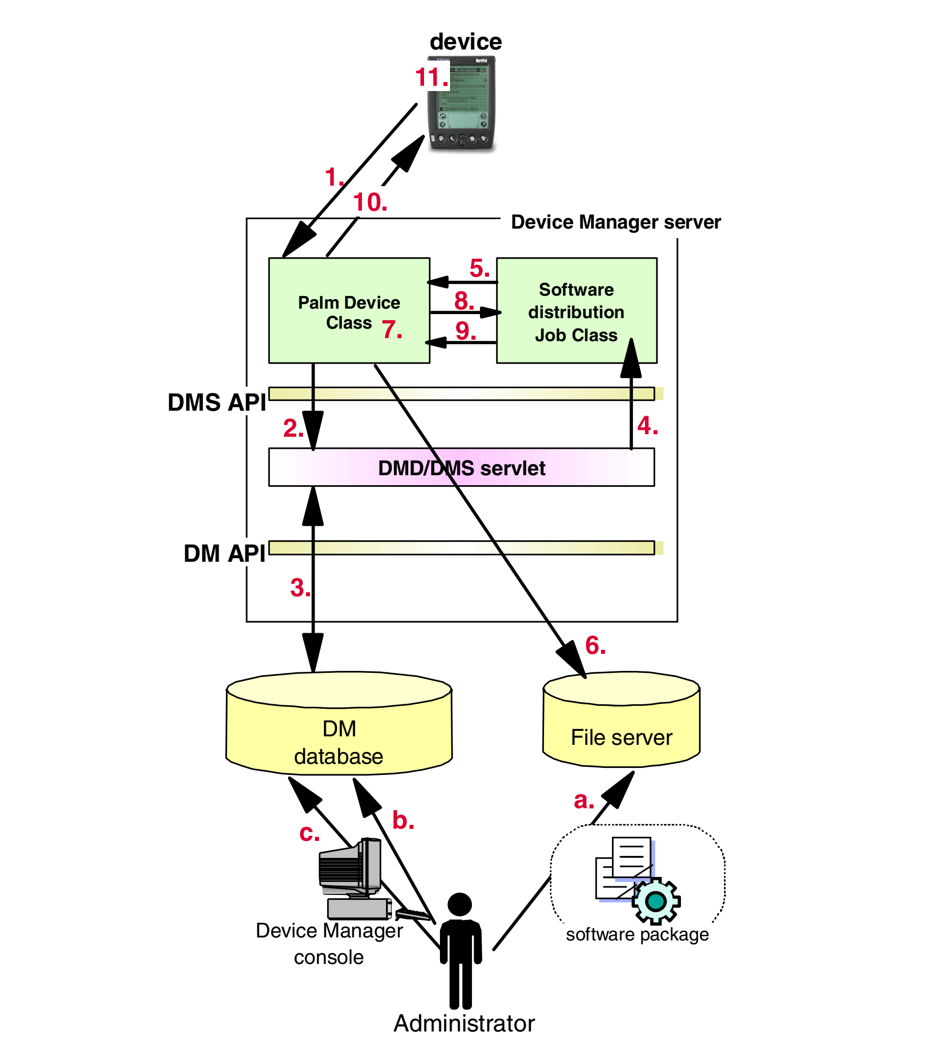


Figure 20. Software distribution job control flow1

Figure 20 illustrates the flow of the software distribution job. Software distribution jobs for Palm device classes are performed as follows:

1. The Device agent sends a RequestJob event to the DeviceCommunication servlet, which is a component of the device class.
2. The device class sends a REQUEST event to the DMS servlet.
3. The DMS servlet checks the DM DB to see if there are any jobs for this device. This time, a software distribution job is found. It obtains information such as the software package URL from the database.
4. The DMS servlet creates an instance of the software distribution job class. It sets the URL of the software package file as a job parameter. Then, it calls the doJob method of the job class.
5. The job class calls getPackageData, which is one of the device class APIs, with software package URL.
6. The device class gets the software package file from the specified URL.
7. The device class reads the software package file and creates the application list.
8. The device class returns the result.
9. The job class calls putPackage (device class API.)
10. The device class sends the application list to the device agent.
11. The agent displays the application list. The user selects some applications and clicks **Install** to install them.

Software distribution jobs for Palm device classes is performed in two steps. In the middle of the job processing, it displays the graphical interface to list the applications to be distributed, or to ask whether to install the software package or not. The format of the application list is different according to the software package definition. For detailed information on software package definition, refer to *Tivoli Internet and Personalized Services Manager Device Manager: Device Plug-in Notes* in Device Manager online documentation. Chapter 7, “Using the DM functions” on page 193, also describes the use of the software package definition.

Some software distribution job classes do not have any graphical interface. Plug-in developers can design the plug-in either to display the graphical interface on the device or not, according to the device characteristics or the system requirements.

After the user chooses to install any of the application, the agent connects to the DM server again as shown in Figure 21.

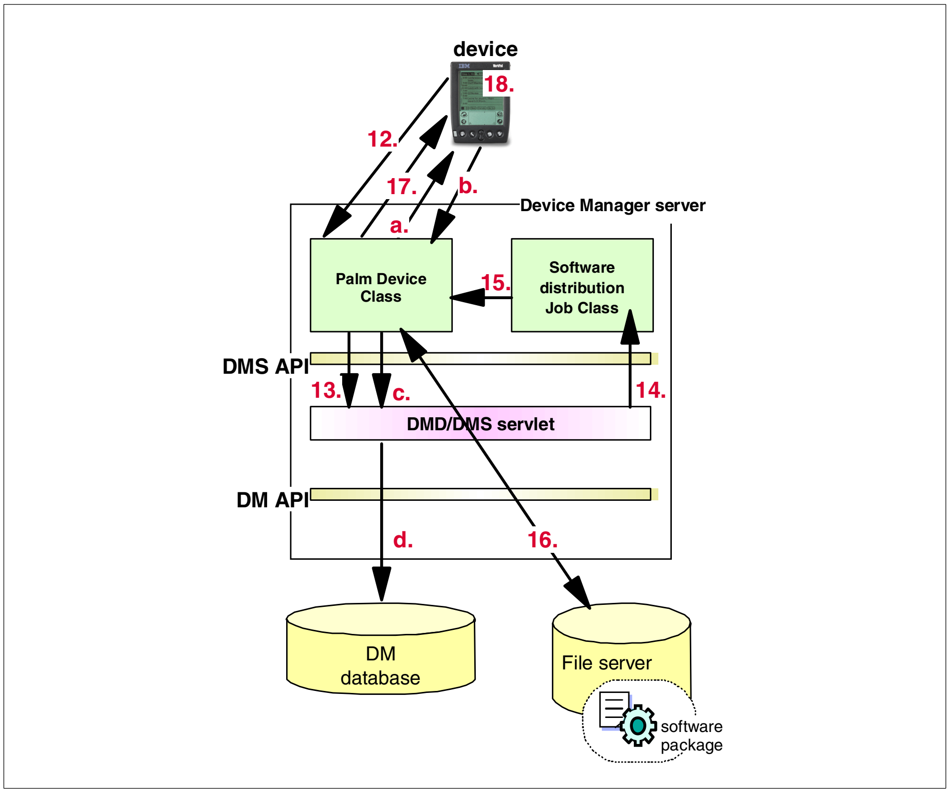


Figure 21. Software distribution job control flow2

The numbered arrows in Figure 21 are explained here:

12.After the user clicks **Install**, the agent sends a Request to the device class, actually DeviceCommunication servlet.

13.DeviceCommunication servlet sends a CONTINUE event to the DMS servlet.

14.The DMS servlet calls the doJob method of the job class again.

15.The job class calls the putPackage (Device class API) again.

16.The device class receives the application ID, file ID and file offset of the file as the parameter, which are sent by the agent. If this is the first time this file is put to the device, the device class gets the file from the URL.

17.The device class sends the specified file from the specified offset.

18.The agent receives the file and converts it if necessary.

Then, the agent resends the ContinueJob event to the server and then repeats the process from 12 to 18 (the numbers of the arrows in Figure 21 on page 51) until all files are installed to the device. When the DM server sends the last cluster of files to the device, it processes the job as shown by arrows a through d in Figure 21 on page 51, to complete the job.

- **Arrow a**: The device class sends the file to the agent, together with the information which indicates this is the last file to send.
- **Arrow b**: After receiving all the files, the agent sends the CompleteJob event to the device class (DeviceCommunication servlet.)
- **Arrow c**: The DeviceCommunication servlet sends the JobComplete event to the DMS servlet.
- **Arrow d**: The DMS servlet updates the progress status of this job and other information related to this job in DM DB.

Now, the software distribution job has successfully completed.

2.4 Summary

This section summarizes the main concepts of this chapter.

2.4.1 Features

Device Manager has the following features:

- Device management service

  Device Manager 1.1, which is a feature of Tivoli Personalized Services Manager, provides the device management service to the other TPSM components and external applications.

  Device Manager uses the relational database as the repository of the device management information. Device management API defines the programming interface for the other components and external applications to access and update the information stored in the Device Manager database. Thanks to this API, other applications can use Device Manager as their job processing engine on the device.

- Scalability and high availability

  Scalability can be archived if the set of DM server machines or TPSM server machines are deployed with front-ended Network Dispatcher.

  Deployed with IBM AIX HACMP, the Device Manager database server can achieve high availability.

- Device support

  Thanks to the plug-in architecture, Device Manager can support various types of devices. The plug-in will be provided for each type of device, and can be developed by device vendors, ISPs, or other developers.

2.4.2 Components

Device Manager consists of the following components:

- Device Manager database

  The central data repository that stores all the device management information.

- Device management server servlet

  Device management server servlet handles requests related to job processing. It also provides the redirect function, thanks to which the devices use only one DM server during the connection, bypassing the front-ended Network Dispatcher.

- Device plug-ins and agents

  Device plug-ins provide the logic that handles device identification, communications, job processing, and high-level management tasks for a particular class of managed devices. Plug-ins consist of device class, job class, and optionally, template. The device agents reside on the device and communicate with the server side plug-ins.

- Device Manager console

  The Win32 based utility which provides the administrators with the graphical interface to administer devices, device classes, jobs and software.

- Device management API

  Device management API provides the programming interface for managing devices, jobs, and related resources in the Device Manager database.

- Device management server API

  Defines the programming interface between device management server servlet (DMS servlet) and plug-ins.

2.4.3 Job processing

Device Manager performs the device management jobs in “pull” style. Generally, the job is processed using the following steps:

1. At the time the device agent connects to the DM server, the server checks the Device Manager database whether the device is already enrolled:
   - If it is found in the DM DB, the server redirects the device directly to the DM server to bypass the front-ended Network Dispatcher.
   - If it is not yet enrolled, the server redirects the device to the Subscription Manager server which enrolls the device and submits a device configuration job as the initial configuration job. Then, it redirects the device to the DM server again.
2. The DM server starts to process the job. It checks the DM DB for any submitted jobs, and if any, initiates the corresponding job class.
3. The job class coordinates to process the job, communicating with the device class.
4. If the job has completed successfully, the DM server updates the DM DB to record the job completion.

Chapter 3. Installation and configuration

TPSM can be deployed by itself, and it also can be deployed as a component of Everyplace Suite. With these two deployments, there are some differences in planning considerations, prerequisites, and installation. Whichever you choose, you should understand the TPSM installation and should consider the system configuration well in advance. This is because TPSM can be configured differently depending on the system requirements, and the installation of TPSM is rather complex.

This chapter summarizes the keys for TPSM installation and configuration planning. For details on planning, refer to the TPSM manual. If you deploy TPSM as an Everyplace Suite component, *WebSphere Everyplace Suite Getting Started v1.1* and *An Introduction to IBM WebSphere Everyplace Suite Version 1.1,* SG24-5995, will help you to understand and plan for your Everyplace Suite system. We also introduce some sample TPSM system configurations, and describe the installation flow of TPSM system installation and configuration tasks.

3.1 Planning

TPSM consists of two main features: the Subscription Manager feature and the Device Manager feature. Taking performance, availability and security into consideration, you need to consider the following five factors.

- TPSM server
  - Subscription Manager server (SM server)
  - Device Manager server (DM server)
- TPSM database server
- File server for software distribution package
- Device Manager console PC
- WebSphere Everyplace Suite environment

This section discusses TPSM system specific considerations and guidelines, and introduces some sample system configurations.

For network configuration, security, database system configuration, and other considerations regarding general network and server configuration, follow the general guidelines or consult each product’s documentation. The following Web site can help you to design the network in a Web environment:

```
http://www.ibm.com/security/library
```

TPSM servers can be flexibly configured according to the system requirements. You can configure DM server and SM server in one server machine, or in different server machines. In addition, you can use front-ended Network Dispatcher to archive scalability.

Considerations

The following items should be considered while planning for TPSM.

- Performance
  - The number of servers required to support the target number of concurrent devices during peak usage.
  - The performance of Web server, Web application server.
- Scalability
  - Whether to use front-ended Network Dispatcher.
  - Whether the servers are used only for Device Manager services, or are also used for other Subscription Manager services.
- High availability
  - Whether to deploy more DM or TPSM servers than required for the target number of devices.

Guidelines

The following guidelines will assist you while planning for your TPSM server.

- In a small system, it is cost effective to have SM server and DM server reside in the same machine.
- Running Device Manager on a dedicated machine allows device support to be scaled independently of other Subscription Manager services, provided all machines are identically configured.
- If you use more than one server for the TPSM server, use IBM Network Dispatcher. For more detailed information on IBM Network Dispatcher, see the Web site:

  `http://www.ibm.com/software/network/dispatcher/`
- If you want to ensure satisfactory operation and response time even if one server is down, use more Device Manager servers than required for the target number of devices.
- Subscription Manager and Device Manager need JDK, Web server, and Web application server as their pre-required software for each. You should tune the performance of Web server and Web application server separately from the SM and DM servers.

3.1.1 TPSM database server

TPSM requires a relational database as its central data repository. TPSM 1.1 supports IBM DB2 UDB v7.1 and Oracle8i as its database system, and supports AIX and Solaris as the platform of the database system. All of the TPSM servers in the system access the centralized TPSM database.

3.1.1.1 Considerations

The following items should be considered while planning for your TPSM database server.

- Performance
  - Whether to reside in the same machine as TPSM server or to reside in the dedicated database server machine.
  - The size of the database.
- High availability
  - Whether or not to use the HACMP solution.

3.1.1.2 Guidelines

The following guidelines will assist you while planning for your TPSM database server.

- By using the IBM AIX HACMP solution, you can archive database high availability. For detailed information about IBM AIX HACMP, refer to the following Web site:

  `http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/hacmp_index.html#V44`

  Also, see 3.5, “References” on page 105 , for more references.
- DM database and SM database reside in the same database, but use different sets of tables. Based on the number of the supported or deployed devices, estimate the size of the DM database. Follow the formula described in Chapter 5, “Device Manager database” on page 145.

  For more information about Subscription Manager database, see *Tivoli Internet and Personalized Services Manager: Planning and Installation*.
- For detailed information on database performance tuning, refer to *Database Performance on AIX in DB2 UDB and Oracle Environments*, SG24-5511.

3.1.2 File server for software distribution package

Device Manager 1.1 supports FILE and HTTP as the connection protocol to the file server which stores the software packages. However, for performance reasons, the FILE protocol is recommended. In this section, we assume that the FILE protocol is used.

File servers can reside either in the same machines as DM servers or in the dedicated file server machines. However, every DM server in the TPSM system should have accessible software packages in the same path, name and context, because the software distribution jobs assume it. That is:

- If you use DM servers also as file servers, you should maintain software package directories so that all servers are identically configured.
- If you use the dedicated file servers, all the mount points on DM servers should be the same path.

3.1.3 Device Manager console PC

In conjunction with an external authentication application, Device Manager console provides the authentication function for the administrators.

There are two authentication method options to choose from. You should decide which to use in advance of TPSM system installation.

- TPSM authentication
- Other authentication

TPSM authentication is the default method in Device Manager, which is provided in conjunction with TPSM Subscription Manager feature.

If you use another method, it must be one that is developed to meet the Device Manager requirements for authentication application. For detailed information on how to develop the authentication application, refer to *Tivoli Personalized Services Manager Device Manager: Developer's Guide* included in TPSM Device Manager 1.1 online documentation.

3.1.4 WebSphere Everyplace Suite environment

Since TPSM interrelates with components of WebSphere Everyplace Suite, it is important that you are familiar with Everyplace Suite components and understand their relation. Detailed information about WebSphere Everyplace Suite can be found in *An Introduction to IBM WebSphere Everyplace Suite Version 1.1,* SG24-5995, and in *WebSphere Everyplace Suite Getting Started v1.1*. The latter can be downloaded from the following Web site:

`http://www.ibm.com/pvc/tech/library.shtml`

As mentioned above, an Everyplace Suite system should be planned well in advance of installation. If you decide to use TPSM, then you should keep in mind single sign-on in addition to other considerations discussed in this section.

3.1.4.1 Single sign-on

Everyplace Suite includes an authentication server component. If TPSM is running in an Everyplace Suite environment, you can take advantage of the authentication server services. When the authentication server is configured as an authentication proxy, it performs user authentication based on HTTP Authenticate headers. It serves as the single point of entry into the Tivoli Personalized Services Manager servers, the network dispatcher, and the Device Manager server. Users who choose to take advantage of this single sign-on capability need to do some extra configuration steps to make sure that redirection from Device Manager to an enrollment server will be done correctly. How to configure Device Manager to support the Authentication Server is described in 3.4.2, “Configuration on WebSphere Application Server” on page 87.

To use the WebSphere Everyplace Suite Authentication Server as an authentication proxy, you must configure the authentication proxy as a reverse proxy, and additional configuration is required to cause the proxy server to route redirection requests for Device Manager. See the chapter on configuration and administration of the Everyplace Suite in *WebSphere Everyplace Suite Getting Started v1.1* for more details.

3.1.4.2 TPSM and other Everyplace Suite components

This section briefly summarizes the relation between TPSM and other Everyplace Suite components.

Everyplace Suite is an integrated, modular suite of software components. Each Everyplace Suite component performs a different function in extending pervasive computing connectivity. Only the components that best provide or extend the services you require need to be installed. Thus, with Everyplace Suite, you need to plan which software components to use well in advance of installation. For detailed information about Everyplace Suite planning, TPSM’s role in the Everyplace Suite environment, see *An Introduction to IBM WebSphere Everyplace Suite Version 1.1,* SG24-5995. Also, *WebSphere Everyplace Suite Getting Started v1.1* helps you to plan and install Everyplace Suite.

TPSM provides services such as active session management, RADIUS server and subscriber and device management to the other components of Everyplace Suite. Everyplace Suite uses LDAP directory service, provided by IBM SecureWay Directory, as its central repository to share information such as user, device and network information. TPSM provisions subscribers' information to the LDAP directory, and all the other components access subscribers' information in the directory.

Everyplace Suite offers a single sign-on environment by using Everyplace Authentication Server. Authentication server uses TPSM RADIUS server to authenticate the subscriber, and updates Active Session Table, which is managed by TPSM Active Session Table server. From the view point of TPSM, TPSM delegates its authentication service to the authentication server. Thus, Everyplace authentication server requires TPSM as its prerequisite software. In addition, Everyplace Suite components, including TPSM, should be configured to support authentication server.

Everyplace Wireless Gateway also uses TPSM RADIUS server and Active Session Table server. The Wireless Gateways are placed in front of authentication server, and access TPSM RADIUS server for authentication. Then, Wireless Gateways update the LDAP directory and Active Session Table, and pass the control to the authentication server. For more detailed information on Everyplace Suite architecture, refer to *An Introduction to IBM WebSphere Everyplace Suite Version 1.1,* SG24-5995.

3.1.5 Sample system configuration

This section shows some sample TPSM system configurations. Please note that these samples do not deeply consider other network elements such as Network Dispatcher and firewalls.

3.1.5.1 Sample 1: simple configuration

The first sample system is configured from one TPSM server machine and one database server machine (see Figure 22.)

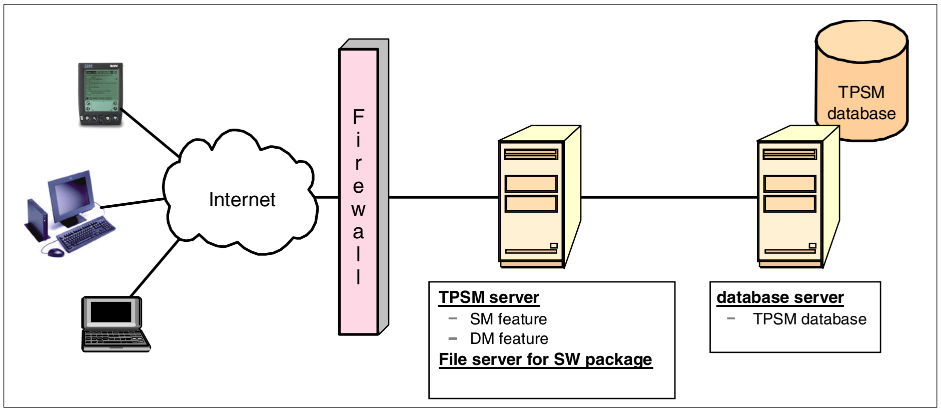


Figure 22. Sample 1: Simple configuration

In this sample, all the TPSM application servers reside in one server machine, and the TPSM database is deployed in the other server machine. In a small system, it is cost effective and easy to maintain.

However, if the number of subscribers and devices or the network traffic is greatly increased to the extent that this system cannot afford to manage them in the future, the whole system should be re-designed.

3.1.5.2 Sample 2: dedicated Device Manager servers

TPSM components can be deployed in multiple server machines. In a large system, you can breakup the network load by dividing TPSM components into multiple boxes. Figure 23 on page 62 is a sample configuration with front-ended Network Dispatcher servers. It deploys SM servers and DM servers in different sets of server machines, and each set is front-ended by the Network Dispatcher. This configuration allows device support to be scaled independently of the Subscription Manager services.

In addition, if you use more DM servers than required for the target number of devices, you can ensure satisfactory operation and response time even if one server is down. With the Network Dispatcher, you can also add other DM or SM server machines to the system, without having to stop the management services.

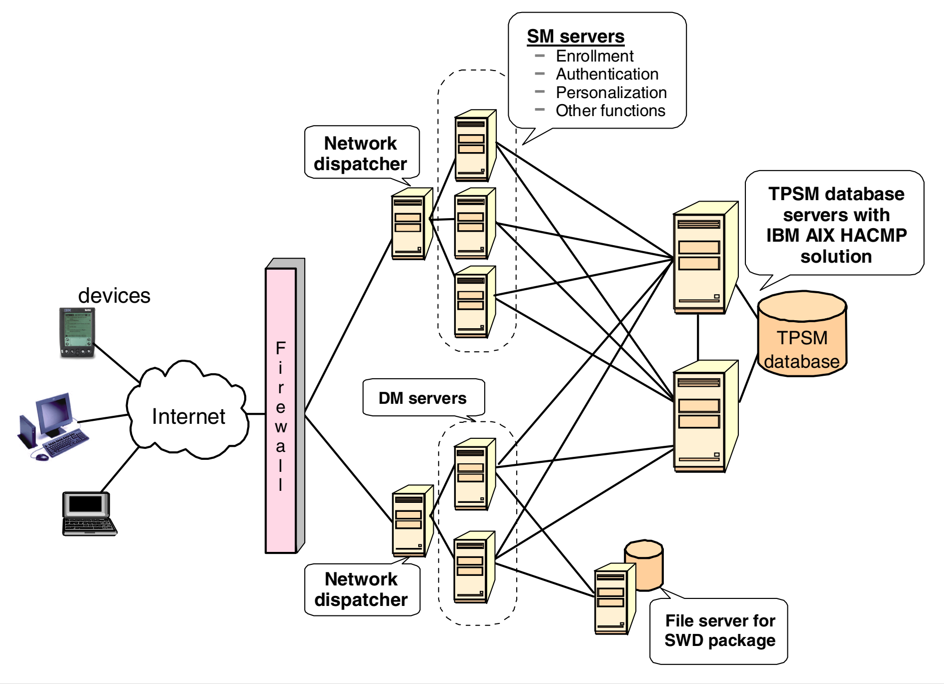


Figure 23. Sample 2: Load balanced TPSM servers

The TPSM database servers in Figure 23 use the IBM AIX HACMP solution to archive the database high availability. The IBM AIX HACMP solution allows customers to automatically detect system failures and recover users, applications and data on backup systems, minimizing downtime to minutes or seconds. In addition, using HACMP for AIX virtually eliminates planned outages, since users, applications and data can be moved to backup systems during scheduled system maintenance.

3.1.5.3 Sample 3: Everyplace Suite deployment

Figure 24 on page 63 is a sample configuration of a WebSphere Everyplace Suite system. To simplify the figure, the relations between Everyplace Suite components are omitted. This deployment has almost all the key components of Everyplace Suite, and could be typical for an enterprise that is looking for a complete solution for pervasive device support along with traditional wired Internet services.

For detailed information on Everyplace Suite system planning, refer to *WebSphere Everyplace Suite Getting Started v1.1*. WebSphere Everyplace Suite: README also provides important information on system planning.

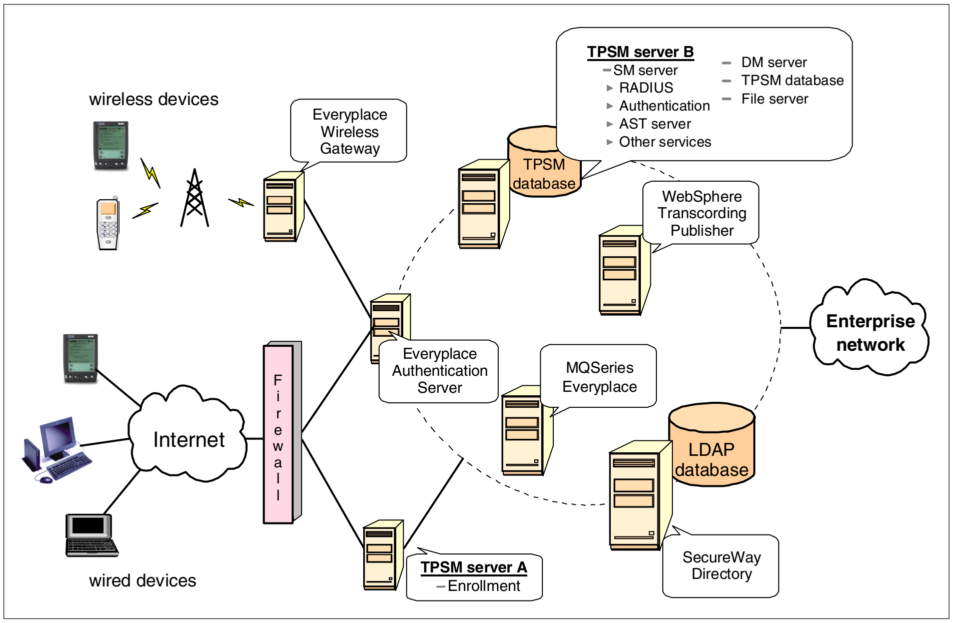


Figure 24. Sample 3: Everyplace Suite deployment

In this deployment, there are two TPSM servers, one is for enrollment service and the other is for other TPSM services. Some SM services, DM server and TPSM database are installed in TPSM server B together.

3.2 Installation flow

This section describes the installation flow overview of the system configuration introduced in 3.1.5, “Sample system configuration” on page 60.

3.2.1 Sample 1: Simple configuration

This section describes the installation flow for the system introduced in 3.2.1, “Sample 1: Simple configuration” on page 63. This system assumes a database server box and a TPSM server box.

Figure 25 on page 65 illustrates the TPSM installation flow. Junctions in the figure mean that branches of it have no installation dependency on each other. For example, you can configure DM server either before creating DM database tables or after creating them, if they have been completed before starting up the DM server.

At first, the database system should be set up for TPSM system. TPSM provides a Database Integration component, which installs the database system and configures it for the TPSM system.

Second, you should configure the runtime environment for the TPSM server. This includes the installation of JDK, HTTP server and WAS. Then, you can install Subscription Manager server and Device Manager server. You can install Device Manager either before Subscription Manager or after it, if you do not start the Device Manager server before starting up the Subscription Manager server. However, we recommend installing Device Manager after the Subscription Manager installation.

After configuring and starting the Device Manager server, install Device Manager console which is used to administer all the device management jobs.

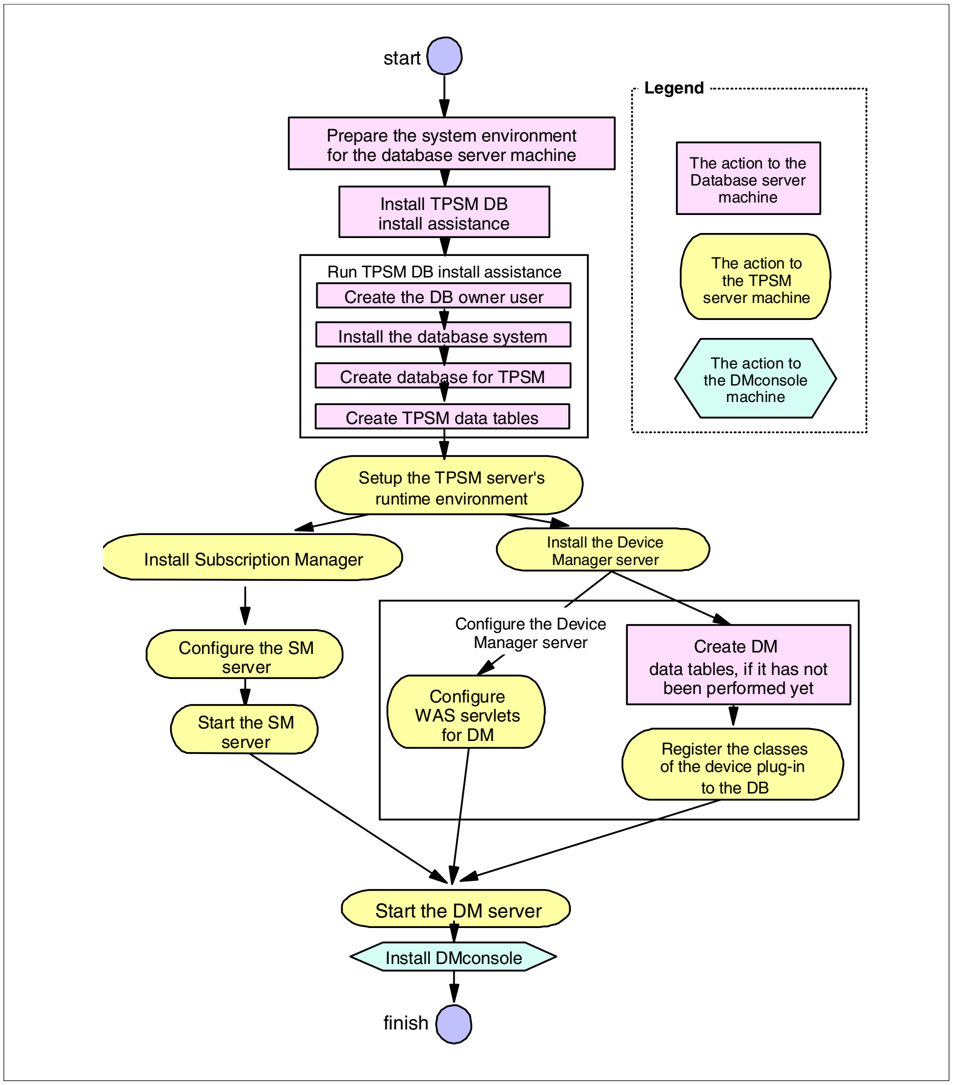

start
Prepare the system environment for the database server machine
Install TPSM DB install assistance
Run TPSM DB install assistance
Create the DB owner user
Install the database system
Create database for TPSM
Create TPSM data tables
Setup the TPSM server's runtime environment
Install Subscription Manager
Install the Device Manager server
Configure the SM server
Start the SM server
Configure the Device Manager server
Configure WAS servlets for DM
Create DM data tables, if it has not been performed yet
Register the classes of the device plug-in to the DB
Start the DM server
Install DMconsole
finish
Legend
The action to the Database server machine
The action to the TPSM server machine
The action to the DMconsole machine

Figure 25. TPSM installation flow overview: Simple configuration

3.2.2 Sample 2: Dedicated Device Manager server

This section describes the installation flow for the system introduced in 3.1.5.2, “Sample 2: dedicated Device Manager servers” on page 61. Figure 26 on page 66 illustrates the installation flow for this configuration.

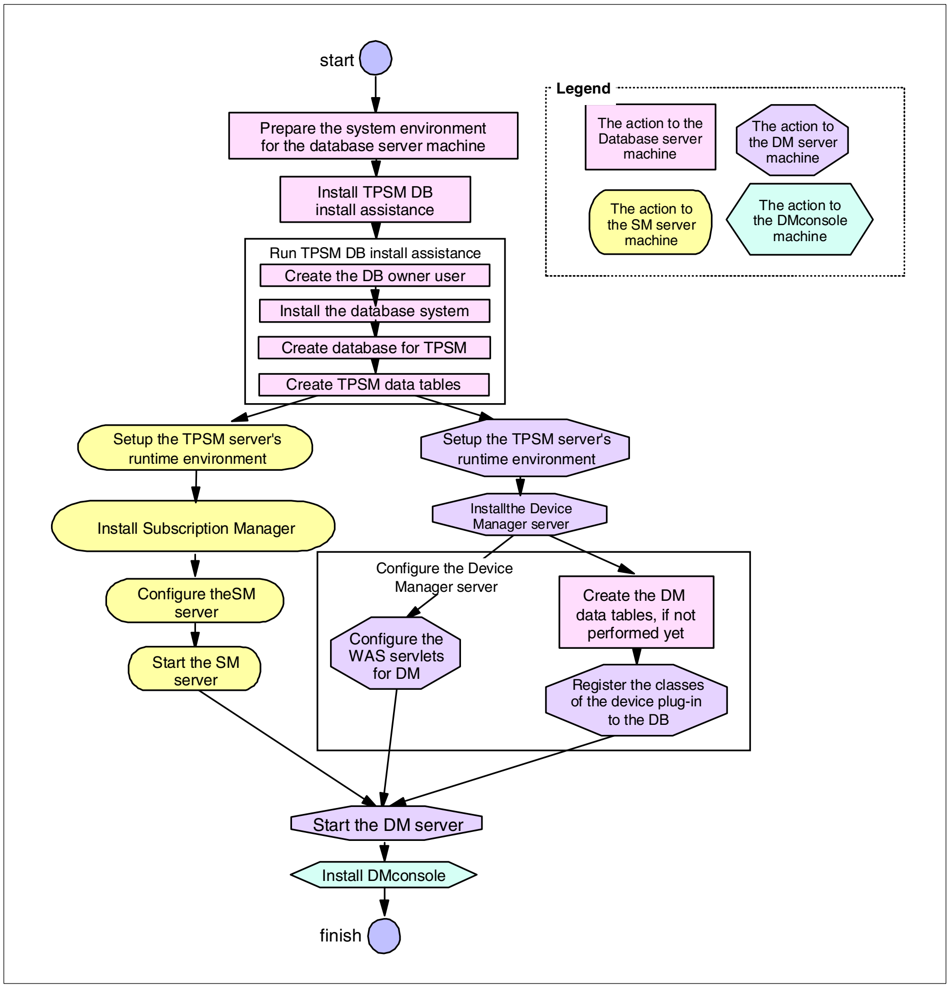

start
Legend
The action to the Database server machine
The action to the DM server machine
The action to the SM server machine
The action to the DMconsole machine
Prepare the system environment for the database server machine
Install TPSM DB install assistance
Run TPSM DB install assistance
Create the DB owner user
Install the database system
Create database for TPSM
Create TPSM data tables
Setup the TPSM server's runtime environment
Install Subscription Manager
Configure theSM server
Start the SM server
Setup the TPSM server's runtime environment
Installthe Device Manager server
Configure the Device Manager server
Configure the WAS servlets for DM
Create the DM data tables, if not performed yet
Register the classes of the device plug-in to the DB
Start the DM server
Install DMconsole
finish

Figure 26. TPSM installation flow overview: Dedicated DM server

The major difference from the installation described in 3.2.1, “Sample 1: Simple configuration” on page 63, is that you have to set up the TPSM runtime environment for both Subscription Manager server and Device Manager server. As described in *Tivoli Personalized Services Manager Device Manager: Planning and Installation*, since Subscription Manager and Device Manager require the same runtime environment, install the same set of software for both servers as the runtime support.

3.2.3 Sample 3: Everyplace Suite deployment

This section describes the installation flow for the system introduced in 3.1.5.3, “Sample 3: Everyplace Suite deployment” on page 62.

We recommend installing Everyplace Suite components in the order illustrated in Figure 27. You can install IBM SecureWay Directory after the other components. However, the installation flow will be simple if you install IBM SecureWay Directory first, because all the other components write their configuration information into the LDAP directory during their installation. As mentioned in 3.1.4.2, “TPSM and other Everyplace Suite components” on page 59, Everyplace Authentication Server and Everyplace Wireless Gateway should be installed after TPSM has been installed, configured and started. For more detailed information on Everyplace Suite installation, refer to *WebSphere Everyplace Suite Getting Started v1.1*.

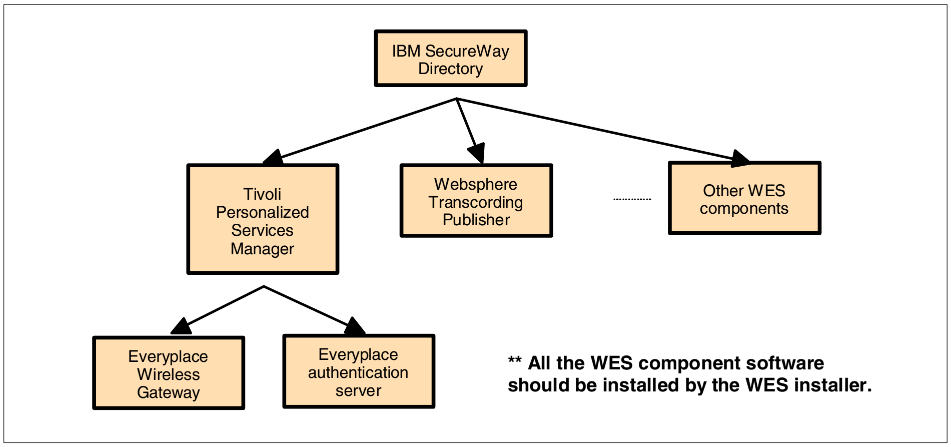

IBM SecureWay Directory
Tivoli Personalized Services Manager
Websphere Transcording Publisher
Other WES components
Everyplace Wireless Gateway
Everyplace authentication server
** All the WES component software should be installed by the WES installer.

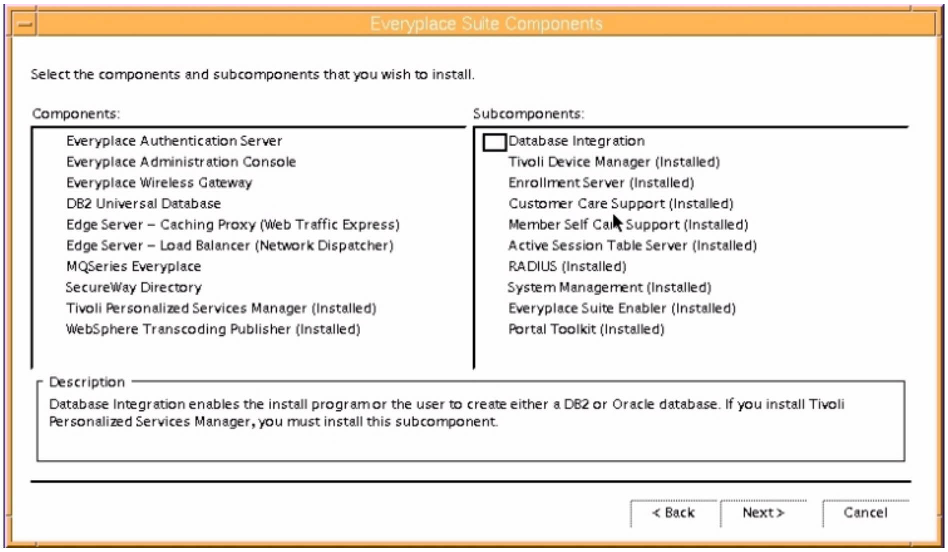
Figure 27. Recommended installation order of Everyplace Suite

Everyplace Suite provides its special installer. In and Everyplace Suite environment, all the component products are installed by Everyplace Suite installer.

First, install IBM SecureWay Directory. Refer to *WebSphere Everyplace Suite Getting Started v1.1* for detailed information on IBM SecureWay Directory installation. While installing SecureWay Directory, write down the following information because they are required in TPSM installation:

- LDAP server name
- The user ID for the SecureWay Directory administrator
- The password for the administrator
- The port number for the directory service

After starting the IBM SecureWay Directory, install TPSM services. The installer provides a window to select the components to be installed as shown in Figure 28. Everyplace Suite installer checks the software dependency in advance of installation. If you are going to install Tivoli Device Manager before Database Integration, it will notify you to install Database Integration first.



Select the components and subcomponents that you wish to install.

Components:

- Everyplace Authentication Server
- Everyplace Administration Console
- Everyplace Wireless Gateway
- DB2 Universal Database
- Edge Server – Caching Proxy (Web Traffic Express)
- Edge Server – Load Balancer (Network Dispatcher)
- MQSeries Everyplace
- SecureWay Directory
- Tivoli Personalized Services Manager (Installed)
- WebSphere Transcoding Publisher (Installed)

Subcomponents:

- □ Database Integration
- Tivoli Device Manager (Installed)
- Enrollment Server (Installed)
- Customer Care Support (Installed)
- Member Self Care Support (Installed)
- Active Session Table Server (Installed)
- RADIUS (Installed)
- System Management (Installed)
- Everyplace Suite Enabler (Installed)
- Portal Toolkit (Installed)

Description

Database Integration enables the install program or the user to create either a DB2 or Oracle database. If you install Tivoli Personalized Services Manager, you must install this subcomponent.

Figure 28. Everyplace installer: Components selection

3.3 Installation

This section describes a sample installation task for a simple system configuration.

3.3.1 System environment

This sample system, shown in Figure 29, uses the minimum configuration of TPSM to run device management services (Subscription Manager enrollment application and Device Manager).

Because this is a simple configuration, one dedicated database server without high availability and a TPSM server without load balancing is used. Subscription Manager enrollment application and Device Manager server, reside on the same machine. The network is private, and a PSTN simulator is used which assumes dial-up users.

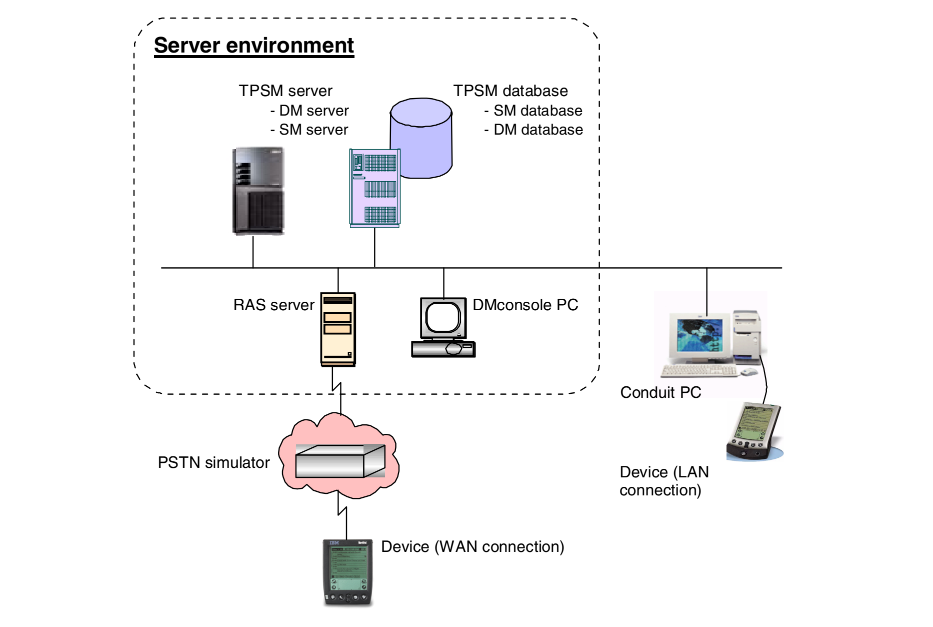


Figure 29. Sample system configuration

Table 4 is the list of software that is installed in the servers shown in Figure 29.

Table 4. List of installed software

Server	Software
TPSM server RS6000 hostname: aix2000.wes.ibm.com	AIX4.3.3 + fix pak2 IBM JDK 1.2.2 (installed with WAS) IBM HTTP Server 1.3.12 (installed with WAS) WAS3.5 Standard Edition Oracle JDBC driver TPSM 1.1 Subscription Manager TPSM 1.1 Device Manager server
Database server Solaris hostname: sol2000.wes.ibm.com	- Solaris7 - Solaris7 recommended patches - patches (106980-13, 107081-22) - Sun JDK1.2.2_05a - Oraclel8i Enterprise Edition 8.1.5 - Subscription Manager database installation script
RAS server	- Windows NT Server 4.0 + SP6a - RAS service

Server	Software
DM console PC	- Windows 98 second edition - Microsoft Internet Explorer 5.5
Conduit PC	- Windows 98 second edition - Palm desktop - TPSM1.1 Palm Cradle Support for Windows
Device	- Device agent for Palm Pilot

3.3.2 Installation tasks

This section describes actual installation steps for the sample system mentioned previously. TPSM installation steps are different depending on the system platform and database product, because their prerequisites are different. If you are going to use other combinations of TPSM and database products than this sample case, refer to *Tivoli Internet and Personalized Services Manager: Planning and Installation*, and confirm the installation steps.

3.3.2.1 TPSM database server installation

This section shows the installation steps of Oracle8i product on Solaris platform. Installation instructions for this combination are not described in detail in *Tivoli Internet and Personalized Services Manager: Planning and Installation*, so these installation steps will help you if you use Sun Solaris and Oracle8i for TPSM database.

TPSM provides the database integration component. In the sequence of this script, it installs Oracle8i database, creates a database for TPSM, and configures the database for Subscription Manager. After this script completes, run the SQL script and create Device Manager database tables. This SQL script creates these tables in the TPSM database.

The following software is used in this case.

- Solaris7
- Solaris7 recommended patches cluster
- Solaris7 patch 106980-13 (Prerequisite of JDK)
- Solaris7 patch 107081-22 (Prerequisite of JDK)
- JDK1.2.2_05a
- Oracle8i Enterprise Edition 8.1.5 for Solaris
- TPSM Oracle database integration

Solaris7 recommended patches cluster and other patches can be downloaded from the Sun Web site at:

http://sunsolve.sun.com/

You can download JDK1.2.2_05a from the following Web site:

http://www.sun.com/software/solaris/java/download.html

Consult the JDK1.2.2_05a Readme file to confirm required Solaris patches, because it requires different sets of patches depending on the system location or other elements. In this case, JDK requires four patches 106980-10, 107636-03, 107081-11, 108376-03. Two patches 107636-03 and 108376-07, are already installed by Solaris7 and recommended patches. Others can be installed according to your system requirements.

The installation tasks are outlined here:

1. Set up the operating system environment.
   a. Install Solaris7. TPSM database requires `/db` as its database system directory, and `/dbfiles` as its database data directory. Be sure to create directories which have those names. It is highly recommended that you create file systems for each and mount them at the time the system starts up. Be sure to point to the proper DNS server so that the host name of the TPSM server can be resolved.
   b. Install the Solaris7 recommended patches cluster, then reboot the system. For detailed installation steps, refer to the Readme file.
2. Install JDK1.2.2_05a, which is the Oracle8i prerequisite software. Follow these instructions:
   a. Install Solaris7 patches, 106980-13 and 107081-22. For detailed installation steps, refer to the Readme file.
   b. Install JDK1.2.2_05a. If you have two versions of JDK co-existing in the system, change the symbolic link /usr/java to point to `/usr/java1.2` after the JDK installation completes. For detailed installation steps, refer to the Readme file.
3. Prepare the system environment for database installation.

   Follow the preconfiguration instructions in the database product's installation manual, which is shipped with Oracle installation image and placed in oracle/docs/solaris.815/a67456.pdf. However, you need not create an Oracle owner user and group, because TPSM Oracle database integration will create them. In the kernel parameter update step, insert the lines shown in Figure 30 on page 72 at the end of the /etc/system file.

```
forceload:      sys/shmsys
forceload:      sys/semsys

set shmsys:shminfo_shmmax=524288000
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=200
set shmsys:shminfo_shmseg=20
set semsys:seminfo_semmni=1000
set semsys:seminfo_semmsl=100
set semsys:seminfo_semmns=210
set semsys:seminfo_semopm=100
set semsys:seminfo_semvmx=32767
```

Figure 30. Example of /etc/system file

Figure 30 shows an example of the /etc/system file of this system, which has 1 GB of RAM. This system has one database instance, and its PROCESSES value in the initispd.ora file is 100.

Two keys are system specific. The value of shmsys:shminfo_shmmax is to the half of the physical memory in bytes. The value of semsys:seminfo_semmns is the result of the following formula. In this, A, B,...,Y, Z are the value of the PROCESSES parameter in the initsid.ora files for each Oracle database. *Z* is the maximum value among them.

$$\sum_{x=A}^{Y} x + (2 \times Z) + (10 \times \text{the number of database})$$

After you edit the file, reboot the system.

4. Install the TPSM Oracle database integration to the database server. It is included in the Subscription Manager Version 1.1 installation. The Solaris version of the TPSM database integration will be installed together with all the other Subscription Manager components. However, the AIX version of the TPSM database integration can be installed independently of other components. Actually, what you install on the Sun server machine to be the TPSM database server is the TPSM 1.1 Subscription Manager feature. On the Sun server machine, perform these steps:

   a. Insert the TPSM installation CD to the CD-ROM drive. Make sure that you are logged in as `root`.

   b. Change the current directory to the CD root. Issue the following command to install the Subscription Manager package named TivTISM:

      ```
      pkgadd -d . TivTISM
      ```

Subscription Manager is installed in the /opt/TivTSM path. TPSM Oracle database integration is placed under the directory.

c. Before you run the database integration program, edit the following files to correct the directories as listed in Table 5:

- /opt/TivTSM/reporting/bin/StartReportTxServer.ksh
- /opt/TivTSM/sysmgmt/bin/StartConsoleTxServer.ksh
- /opt/TivTSM/install/etc/rc.webservers
- /opt/TivTSM/install/etc/rc.txservers
- /opt/TivTSM/install/etc/rc.websphere
- /opt/TivTSM/radius/bin/start_radius.ksh
- /opt/TivTSM/radius/bin/reload_radius.ksh
- /opt/TivTSM/radius/conf/radius-acct.cf
- /opt/TivTSM/radius/conf/radius-auth.cf

Change the five directory names and variables in Table 5 to the ones listed in the right-hand column of the table.

Table 5. Update the following directory names

Before	After
/usr/jdk_base	/usr/java
/usr/TivTSM	/opt/TivTSM
/usr/netscape	/opt/netscape
/var/adm/logs	/var/adm/log
LIBPATH	LD_LIBRARY_PATH

5. On the local machine, run the following commands:

```
export DISPLAY=:0.0
xhost +
```

The following message should be displayed:

```
Access control should be disabled. Clients can connect from any host.
```

If you see the following message:

```
1346-217 xhost: must be on local machine to enable or disable access
control
```

an error has occurred and you must correct the xhost issues before you continue.

6. Run the TPSM Oracle database integration. This script creates schema owner user, installs the database software, configures the database environment, and sets up the Subscription Manager database. The installation tasks are:

a. Insert the Oracle 8 installation CD into the CD-ROM drive. The database integration program requires you to mount the Oracle 8 installation CD on `/cdrom/oracle/`.

b. Change the current directory to `/opt/TivTSM/install/db/oracle`, and run the following commands to set the system environment:

```
./TSMOracle8i
```

Follow the instructions displayed by the program. It may take an hour or more because it will install the Oracle database system, create the database schema, and perform other configuration tasks.

For detailed TPSM database installation tasks, see *Tivoli Internet and Personalized Services Manager: Planning and Installation*. Also, see your database product's documentation for system prerequisites and preparation.

3.3.2.2 TPSM server installation

After you have installed the TPSM database, you can set up the TPSM server. First, you must install runtime support software. Then, install Subscription Manager and Device Manager respectively. These installation instructions assume an AIX version of the installation.

Installing runtime support software

First, install the AIX4.3.3 operating system. TPSM 1.1 requires AIX PTF2 and some AIX support software. TPSM 1.1 also requires Java_dev2.rte 1.2.2.8, Web server and WebSphere Application Server 3.5 Standard Edition. However, Java_dev2.rte and IBM HTTP Server 1.3.12 will be installed together with WAS v3.5, because the installer is designed to perform the installation for them.

In summary, install the runtime support software as explained in the following steps. You can find detailed information on the installation instructions in the documentation for each product.

1. Install IBM AIX4.3.3.
2. Install AIX PTF2.
3. Install the following AIX support software if they are not currently installed. All of them are included in the AIX product CD.
    - X11.adt 4.3.3.0
    - X11.base.lib
    - X11.base.rte
    - AIX smitty

4. Install WebSphere Application Server 3.5 Standard Edition. Be sure to choose the following options. This also installs Java_dev2.rte and IBM HTTP Server 1.3.12.
   a. On the Install Options panel, choose **Custom Installation**.
   b. On the Choose Application Server Components panel, select **IBM HTTP Server**.
   c. On the Choose Application Server Components panel, select **IBM HTTP Server plug-in**.
   d. On the Database Options panel, select **InstantDB**.

Installing TPSM

After you install the runtime support software, install the Subscription Manager server. Detailed installation tasks are described in *Tivoli Internet and Personalized Services Manager: Planning and Installation*. After you install Subscription Manager, install Device Manager as explained in the following steps. Please note that no installation logs are created by the DMS installer. We recommend that you redirect the standard output to a file.

1. Insert the product CD into the CD-ROM drive, and change the current directory to the CD-ROM root. The Device Manager installation script, `dms_install.sh`, should be there.
2. Run the `dms_install.sh` script. You are asked to type the path of the JDBC driver. Since this Device Manager installation is for an AIX machine and the database system is Oracle, the path to type here is `/usr/lpp/TivDMS/doc/classes12_01.zip`. After the installation completes, copy the JDBC driver to the path above.

   The path you should specify is:

   - **AIX + DB2 UDB**: `/usr/lpp/TivDMS/doc/db2java.zip`
   - **AIX + Oracle**: `/usr/lpp/TivDMS/doc/classes12_01.zip`
   - **Solaris + DB2 UDB**: `/opt/TivDMS/doc/db2java.zip`
   - **Solaris + Oracle**: `/opt/TivDMS/doc/classes12_01.zip`

   Figure 31 on page 76 shows an example of a screen from the Device Manager installation.

```
aix2000[/tmp/dms]#./dms _install.sh

---------------------------------------------------------------------------------

The Oracle8i JDBC thin-client driver for JDK 1.2.x (classes12_01.zip) OR
the DB/2 UDB JDBC thin-client driver for JDK 1.2.x (db2java.zip) is required,
and must exist in the /usr/lpp/TivDMS/doc directory after installation.

---------------------------------------------------------------------------------

Enter the path and filename of JDBC driver: /usr/lpp/TivDMS/doc/classes12_01.zip

>>> JDBC driver /usr/lpp/TivDMS/doc/classes12_01.zip was not found !!!

The Oracle8i JDBC thin-client driver (classes12_01.zip).
The DB/2 UDB JDBC thin-client driver (db2java.zip).
One of the above is required and must be copied to the
/usr/lpp/TivDMS/doc directory after installation.
```

Figure 31. Example screen of Device Manager installation

3. Enter the following information as shown in Figure 32 . Since The TPSM Subscription Manager database installation scripts create the same database entries as prompted here by default, you can simply press Enter to accept the default settings for the database.
   - The host name of the database machine
   - DB user ID
   - DB user’s password
   - DB connect port number
   - DB connect name

```
==========================================================================
Enter the hostname of the machine that the Database was installed on: sol2000
Enter DB user ID (default use ID = stage_user): (press Enter)
DB user ID is stage_user
Enter DB password (default password = oracle): (press Enter)
DB password is oracle
Enter DB connect port number (default port number = 1521): (press Enter)
DB connect port is 1521
Enter DB connect name (default DB name = ispb) : (press Enter)
DB name is ispb
----------------------------------------------------------------------
```

Figure 32. Default settings for the TPSM database

4. You can confirm your previous input, as shown in Figure 33. If you made a mistake, you can reply with an “n”, and return to the previous screen.

```
The following data will be used in Transaction.properties file:

DB hostname:  sol2000
DB user ID:  stage_user
DB password:  oracle
DB connect port:  1521
DB name:  ispb
----------------------------------------------------------------------

Are they correct (y/n)? y
```

Figure 33. Data entry confirmation screen

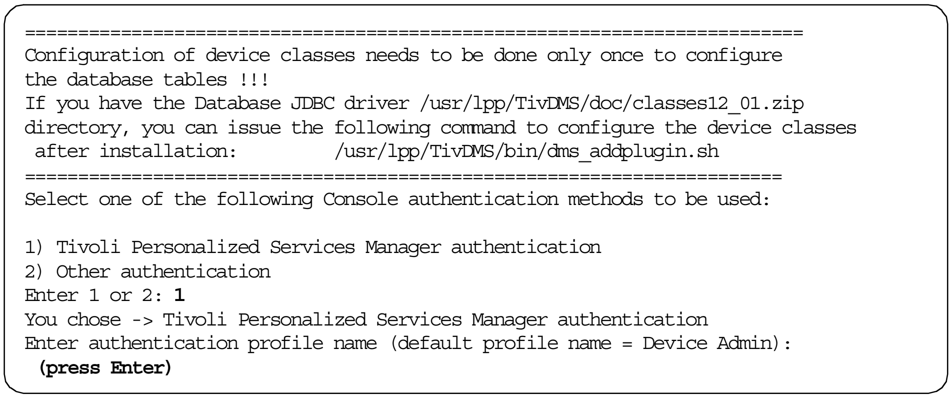
5. Specify the port number used by the Device Manager Server servlet as shown in Figure 34. In some cases, this value is 80, but if you only use an SSL connection, specify `443` as the port number.

```
=========================================================================
Specify the port number used by Device Manager Server Servlet
(default port number = 80): (press Enter)

---------------------------------
Port number is 80
---------------------------------
```

Figure 34. DMS servlet port number specification

6. Select the authentication method for administrators. As shown in Figure 35 on page 78, choose TPSM authentication (**1**) if you have not developed the other authentication application to meet the Device Manager requirements for the authentication application. This method can also be changed later.

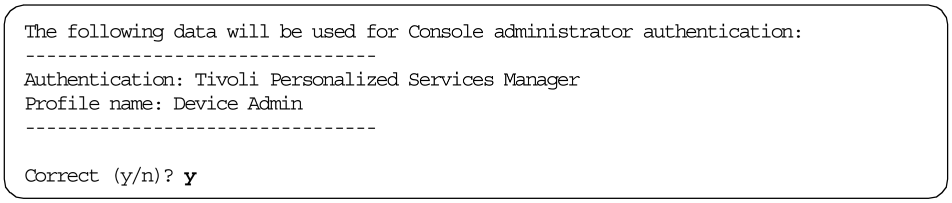
   Next, specify the authentication profile. If you have not specially created the profile for administrators, simply press Enter and accept the default. The profile used for the administrator authentication can be changed later.



```
=========================================================================
Configuration of device classes needs to be done only once to configure
the database tables !!!
If you have the Database JDBC driver /usr/lpp/TivDMS/doc/classes12_01.zip
directory, you can issue the following command to configure the device classes
 after installation:         /usr/lpp/TivDMS/bin/dms_addplugin.sh
=======================================================================
Select one of the following Console authentication methods to be used:

1) Tivoli Personalized Services Manager authentication
2) Other authentication
Enter 1 or 2: 1
You chose -> Tivoli Personalized Services Manager authentication
Enter authentication profile name (default profile name = Device Admin):
 (press Enter)
```

Figure 35. Console authentication method specification panel

7. You can confirm your input as shown in Figure 36. If you made a mistake, you can reply with an “n” and return to the previous screen.

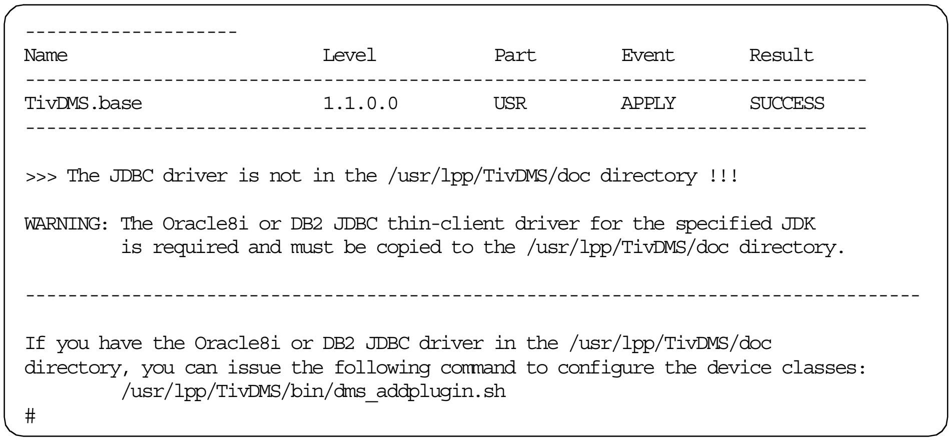


```
The following data will be used for Console administrator authentication:
---------------------------------
Authentication: Tivoli Personalized Services Manager
Profile name: Device Admin
---------------------------------

Correct (y/n)? y
```

Figure 36. Authentication method confirmation panel

8. Once all the information has been entered, the installation script starts copying the files.

9. The message shown in Figure 37 is displayed, and the installation is complete.



```
--------------------
Name                        Level           Part        Event       Result
-------------------------------------------------------------------------------
TivDMS.base                 1.1.0.0         USR         APPLY       SUCCESS
-------------------------------------------------------------------------------

>>> The JDBC driver is not in the /usr/lpp/TivDMS/doc directory !!!

WARNING: The Oracle8i or DB2 JDBC thin-client driver for the specified JDK
         is required and must be copied to the /usr/lpp/TivDMS/doc directory.

------------------------------------------------------------------------------------

If you have the Oracle8i or DB2 JDBC driver in the /usr/lpp/TivDMS/doc
directory, you can issue the following command to configure the device classes:
         /usr/lpp/TivDMS/bin/dms_addplugin.sh
#
```

Figure 37. TPSM installation complete panel

10. Copy the JDBC driver to the DMS doc path `/usr/lpp/TivDMS/doc` (AIX) or `/opt/TivDMS/doc` (Solaris).

Since this configuration uses an AIX system as the Device Manager and Oracle 8.1.5 for the database system, copy the Oracle8i JDBC driver classes12_01.zip to the `/usr/lpp/TivDMS/doc` path.

3.3.2.3 Creating Device Manager data tables

When the TPSM database integration has been installed to the database server, Device Manager data tables can be created. However, this step can be performed on the database server after Device Manager has been installed. The Device Manager database installation scripts are provided in the product path `/usr/lpp/TivDMS/bin` (AIX) or `/opt/TivDMS/bin` (Solaris), which are created in the sequence of Device Manager installation. There are six script files for the Solaris version of DM database:

- create_DMS_ora.sql
- createDMSSeqs_ora.sql
- createDMSSyns_ora.sql
- createDMSTables_ora.sql
- createDMSTriggers_ora.sql
- createDMSViews_ora.sql

If you use DB2 UDB as the TPSM database system, the following five scripts are used:

- create_DMS_db2.sql
- createDMSSyns_db2.sql

- createDMSTables_db2.sql
- createDMSTriggers_db2.sql
- createDMSViews_db2.sql

1. Copy the SQL scripts to the database server from the Device Manager server. These scripts assume that the following Device Manager default settings are used:
   - **Database schema user**: stage_user
   - **Tablespace name**: ISPB_DATA
   - **Database schema owner**: stage_master

   If you are using other names than those above, edit the SQL scripts to meet your system environment before you execute them.
2. Login to the database server as a user who has appropriate permission to run `sqlplus`.
3. Change the current directory to the path in which the SQL scripts exist.
4. Run `sqlplus`. Login as `stage_user`, who is the owner of the Device Manager database.
5. Execute the following script. This script calls others and creates all the tables and other database elements.

   ```
   sqlplus> @create_DMS_ora
   ```

   This script calls all the other scripts. These scripts drop existing tables or other elements first, then creates new ones. All of these attempts are logged to the create_DMS_ora.log file.
6. Exit sqlplus.

Next, register device plug-in classes to the database. This procedure is explained in 3.4.1, “Registering the device plug-in classes” on page 81.

3.4 Configuration

After you install Device Manager, register the device classes and job classes to the Device Manager database, and configure the WebSphere Application Server.

If you install the Device Manager from Everyplace Suite installer, it will perform most of the configuration on WAS for Device Manager. Perform the configuration on WAS, set only one parameter, and register the classes in the Everyplace Suite deployment.

In summary, for a non-Everyplace Suite environment, you need to complete the following required configuration tasks:

- Register the device plug-in classes. Follow the instructions in 3.4.1, “Registering the device plug-in classes” on page 81.
- Configure WAS for Device Manager. Follow the instructions in 3.4.2, “Configuration on WebSphere Application Server” on page 87.

In an Everyplace Suite environment, you need to complete the following required configuration tasks:

- Register the device plug-in classes. Follow the instructions in 3.4.1, “Registering the device plug-in classes” on page 81.
- If you use Everyplace Authentication Server in the Everyplace Suite environment, set a `authProxyDmsUrl` parameter for the DeviceManagementServer servlet on the WAS administrator console. The actual configuration instructions are in 3.4.2, “Configuration on WebSphere Application Server” on page 87.

The following section explains how to register the classes, configure WAS for Device Manager, and start and shut down the DM server.

3.4.1 Registering the device plug-in classes

Before you start the Device Manager server, you need to configure the Device Manager database. To configure the database, build the database tables and register the device classes and job classes for specific device classes to the Device Manager database. This section describes the tasks needed to register the classes to the database. How to create the tables in TPSM database is described in 3.3.2.3, “Creating Device Manager data tables” on page 79.

There are two ways to register the classes to the database. One is to use the `dms_addplugin.sh` script, which provides interactive installation. The other is to use the `devclasscfg.sh` and `jobclasscfg.sh` scripts. By creating a shell script file including the scripts, you can execute unattended plug-in installation.

3.4.1.1 Registering using dms_addplugin.sh

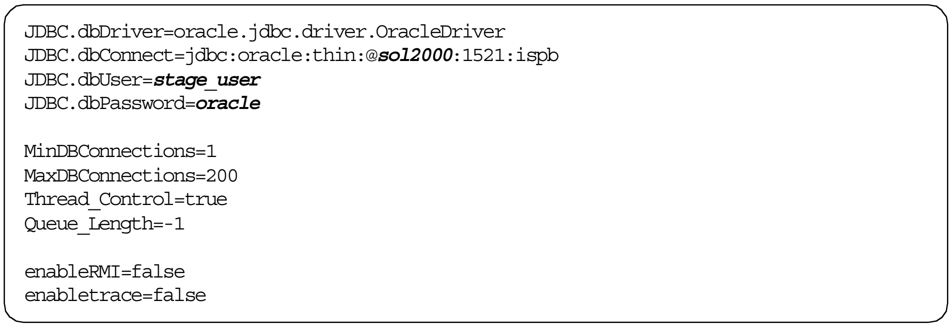
Device Manager provides the `dms_addplugin.sh` script file, which is located in the path `/usr/lpp/TivDMS/bin` (AIX) or `/opt/TivDMS/bin` (Solaris). This script file performs the registration of built-in device classes and job classes interactively. Using this script, you do not have to be concerned with the detailed options actually required for the registration. For example, it will register the set of device parameters to the database.

You should gather the following information before you start the script:

- Device plug-ins you want to register.
- The device enrollment URL for each device plug-in, if possible. This can be registered later using the Device Manager console.

To register device plug-ins with `dms_addplugin.sh`, complete the following steps. This case assumes that only the Palm device plug-in is registered.

1. Before you start `dms_addplugin.sh`, edit the Transaction.properties file located in `/usr/lpp/TivDMS/Class` (AIX) or `/opt/TivDMS/Class` (Solaris). As shown in Figure 38, change the words after the "@" character in the second line to the database server name (shown in bold italic), the value of JDBC.dbUser in the third line to the schema user, and the value of JDBC.dbPassword in the forth line to the schema user's password.



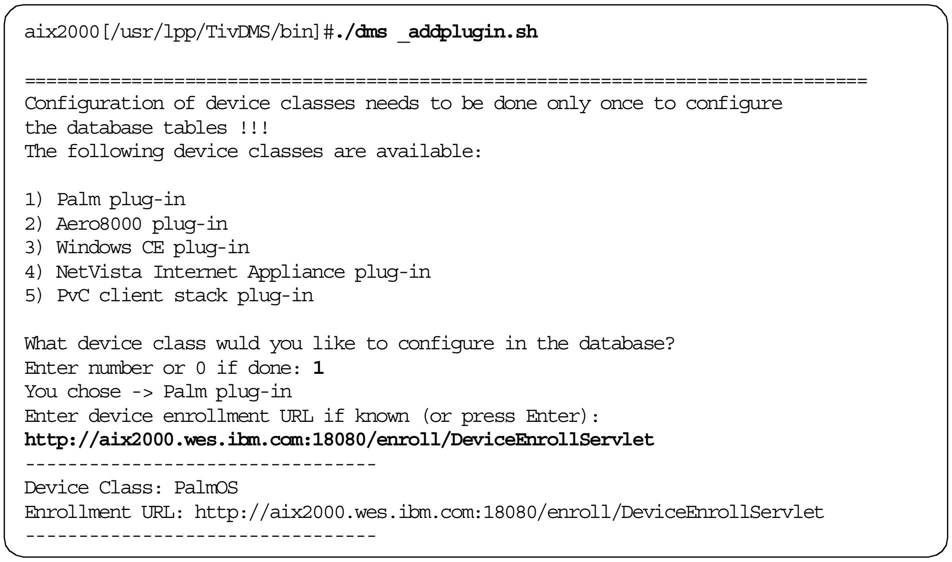
```
JDBC.dbDriver=oracle.jdbc.driver.OracleDriver
JDBC.dbConnect=jdbc:oracle:thin:@sol2000:1521:ispb
JDBC.dbUser=stage_user
JDBC.dbPassword=oracle

MinDBConnections=1
MaxDBConnections=200
Thread_Control=true
Queue_Length=-1

enableRMI=false
enabletrace=false
```

Figure 38. Editing the Transaction.properties file

2. The `dms_addplugin.sh` script is placed in the path `/usr/lpp/TivDMS/bin` (AIX) or `/opt/TivDMS/bin` (Solaris). Change the directory and issue the following command as the root:

   ```
   ./dms_addplugin.sh
   ```

3. A list of device plug-ins is displayed, and you are prompted to enter the number of the device plug-in that you want to configure as shown in Figure 39.



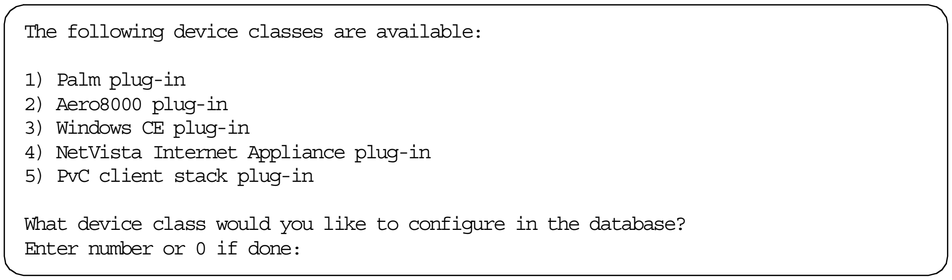
```
aix2000[/usr/lpp/TivDMS/bin]#./dms _addplugin.sh

==============================================================================
Configuration of device classes needs to be done only once to configure
the database tables !!!
The following device classes are available:

1) Palm plug-in
2) Aero8000 plug-in
3) Windows CE plug-in
4) NetVista Internet Appliance plug-in
5) PvC client stack plug-in

What device class wuld you like to configure in the database?
Enter number or 0 if done: 1
You chose -> Palm plug-in
Enter device enrollment URL if known (or press Enter):
http://aix2000.wes.ibm.com:18080/enroll/DeviceEnrollServlet
---------------------------------
Device Class: PalmOS
Enrollment URL: http://aix2000.wes.ibm.com:18080/enroll/DeviceEnrollServlet
---------------------------------
```

Figure 39. Device plug-in registration selection panel

4. Type `1` to register a Palm plug-in.

   Figure 39 also shows a prompt, which asks you to enter the device enrollment URL, if known.

5. The panel shown in Figure 40 is displayed. If you want to register other plug-ins, type the corresponding number. Again, you are prompted for the enrollment URL for the plug-in.

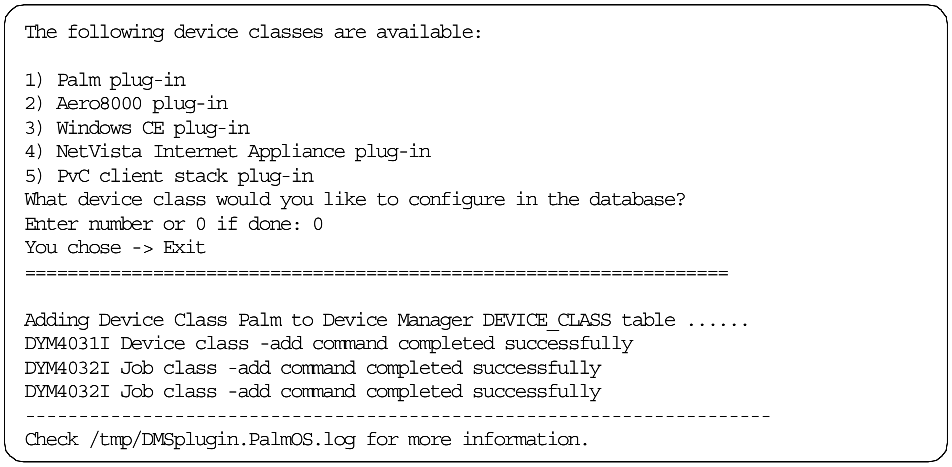


```
The following device classes are available:

1) Palm plug-in
2) Aero8000 plug-in
3) Windows CE plug-in
4) NetVista Internet Appliance plug-in
5) PvC client stack plug-in

What device class would you like to configure in the database?
Enter number or 0 if done:
```

Figure 40. The device plug-in selection panel

6. If you have finished selecting all the plug-ins you want to register, enter `0` in the prompt to exit, and add the classes to the Device Manager database.

7. Registration starts. You can confirm the installation results by checking the log files created in the /tmp directory as shown in Figure 41. They have names such as DMSplugin.*plugin_name*.log.



```
The following device classes are available:

1) Palm plug-in
2) Aero8000 plug-in
3) Windows CE plug-in
4) NetVista Internet Appliance plug-in
5) PvC client stack plug-in
What device class would you like to configure in the database?
Enter number or 0 if done: 0
You chose -> Exit
==================================================================

Adding Device Class Palm to Device Manager DEVICE_CLASS table ......
DYM4031I Device class -add command completed successfully
DYM4032I Job class -add command completed successfully
DYM4032I Job class -add command completed successfully
----------------------------------------------------------------------
Check /tmp/DMSplugin.PalmOS.log for more information.
```

Figure 41. Device plug-in registration completion panel

3.4.1.2 Registering by using devclasscfg.sh and jobclasscfg.sh

Device Manager provides `devclasscfg.sh` and `jobclasscfg.sh` scripts, which are located in `/usr/lpp/TivDMS/bin` (AIX) or `/opt/TivDMS/bin` (Solaris). They perform general management tasks for device classes and job classes on the server.

If you want to register device plug-ins using an unattended method, you need to create a shell script file that includes the `devclasscfg.sh` and `jobclasscfg.sh` command lines, and place it in the appropriate path. Figure 42 shows a sample script file named `pluginconfig.sh`, which assumes the current directory is `/usr/lpp/TivDMS/bin` (AIX) or `/opt/TivDMS/bin` (Solaris). If you want to place the shell script file in the other directory, be sure to set the environment or change the command line so that they can be run properly.

The syntax and more details about the use of `devclasscfg.sh` and `jobclasscfg.sh` are described in Chapter 4, “Administration” on page 109.

```
#!/bin/ksh
#

./setpath.sh
#############################################
############### pluginconfig.sh #############
#############################################

#############################################
############# Aero 8000 plug-in #############
#############################################

#### register Aero 8000 device class ####
./devclasscfg.sh -add Aero8000 -class com.tivoli.dms.plugin.win.wince.hpcpro
.aero8000.Aero8000
-enroll http://aix2000.wes.ibm.com:18080/enroll/DeviceEnrollServlet
-template Aero8000.template

#### register software distribution job class for Aero8000 plug-in ####
./jobclasscfg.sh -add -type SW_DIST -javaclass
com.tivoli.dms.plugin.base.rdmi.RdmiSoftwareDistributionJob
-deviceclass Aero8000

#### register device configuration job class for Aero8000 plug-in ####
./jobclasscfg.sh -add -type DEVICE_CFG -javaclass
com.tivoli.dms.plugin.win.wince.hpcpro.aero8000.Aero8000DeviceConfigurationJob
-deviceclass Aero8000

#############################################
############# Windows CE plug-in ############
#############################################

#### register WindowsCE device class ####
./devclasscfg.sh -add Wince -class com.tivoli.dms.plugin.win.wince.Wince
-enroll http://aix2000.wes.ibm.com:18080/enroll/DeviceEnrollServlet
-template Wince.template

#### register software distribution job class for WindowsCE plug-in ####
./jobclasscfg.sh -add -type SW_DIST -javaclass
com.tivoli.dms.plugin.base.rdmi.RdmiSoftwareDistributionJob
-deviceclass Wince

#### register device configuration job class for WindowsCE plug-in ####
./jobclasscfg.sh -add -type DEVICE_CFG -javaclass
com.tivoli.dms.plugin.base.rdmi.RdmiDeviceConfigurationJob
-deviceclass Wince
### to be continued to the next page
```

Figure 42. Sample pluginconfig.sh script file (Part 1 of 2)

```
### continued from the previous page

#############################################
############### Palm plug-in ################
#############################################

#### register Palm device class ####
./devclasscfg.sh -add Palm -class com.tivoli.dms.plugin.palm.Palm -enroll
http://aix2000.wes.ibm.com:18080/enroll/DeviceEnrollServlet
-template Palm.template

#### register software distribution job class for Palm plug-in ####
./jobclasscfg.sh -add -type SW_DIST -javaclass
com.tivoli.dms.plugin.palm.PalmSoftwareDistributionJob -deviceclass Palm

#### register device configuration job class for Palm plug-in ####
./jobclasscfg.sh -add -type DEVICE_CFG -javaclass
com.tivoli.dms.plugin.palm.PalmDeviceConfigurationJob -deviceclass Palm


#############################################
################ IAD plug-in ################
#############################################

#### register IAD device class ####
./devclasscfg.sh -add Iad -class com.tivoli.dms.plugin.pvc.iad.Iad -enroll
http://aix2000.wes.ibm.com:18080/enroll/DeviceEnrollServlet
-template Iad.template

#### register software distribution job class for IAD plug-in ####
./jobclasscfg.sh -add -type SW_DIST -javaclass
com.tivoli.dms.plugin.base.rdmi.RdmiSoftwareDistributionJob -deviceclass Iad

#### register device configuration job class for IAD plug-in ####
./jobclasscfg.sh -add -type DEVICE_CFG -javaclass
com.tivoli.dms.plugin.base.rdmi.RdmiDeviceConfigurationJob -deviceclass Iad

#### register restpage management job class for IAD plug-in ####
./jobclasscfg.sh -add -type RESTPAGE_MGMT -javaclass
com.tivoli.dms.plugin.pvc.iad.IadSampleRestPageJob -deviceclass Iad
```

Figure 43. Sample pluginconfig.sh script file (Part 2 of 2)

Complete the following instructions to register the classes using `devclasscfg.sh` and `jobclasscfg.sh`:

1. Before you run the `pluginconfig.sh` script, edit the Transaction.properties file located in `/usr/lpp/TivDMS/Class` (AIX) or `/opt/TivDMS/Class` (Solaris). Change the words after the “@” character in the second line to the database server name (shown in bold italic), the value of JDBC.dbUser in the third line to the schema user, and the value of JDBC.dbPassword in the fourth line to the schema user’s password.

```
JDBC.dbDriver=oracle.jdbc.driver.OracleDriver
JDBC.dbConnect=jdbc:oracle:thin:@sol2000:1521:ispb
JDBC.dbUser=stage_user
JDBC.dbPassword=oracle

MinDBConnections=1
MaxDBConnections=200
Thread_Control=true
Queue_Length=-1

enableRMI=false
enabletrace=false
```

Figure 44. Editing the Transaction.properties file

2. Create a shell script file to register the classes. The previous sample script can be used as a guide.
3. Run the script.

3.4.2 Configuration on WebSphere Application Server

After you install DMS, you should configure the application server on WebSphere Application Server. Follow these steps:

1. Open the administration client console of WebSphere Application Server using the following instructions:

   a. As shown in Figure 45 on page 88, change the directory to `/usr/WebSphere/AppServer/bin` (AIX) or `/opt/WebSphere/AppServer/bin` (Solaris).

   b. To launch WebSphere, issue the command:

   ```
   ./startupServer.sh &
   ```

   This will take several minutes to come up.

   c. To see the log file of WAS, issue the command:

   ```
   tail -f /usr/IBMWebAS/logs/tracefile
   ```

   d. Wait for the message “A WebSphere Administration server open for e-business” to appear. Then, exit the tail command by pressing Ctl + c.

   e. To launch the administration GUI, issue the command:

   ```
   ./adminclient.sh &
   ```

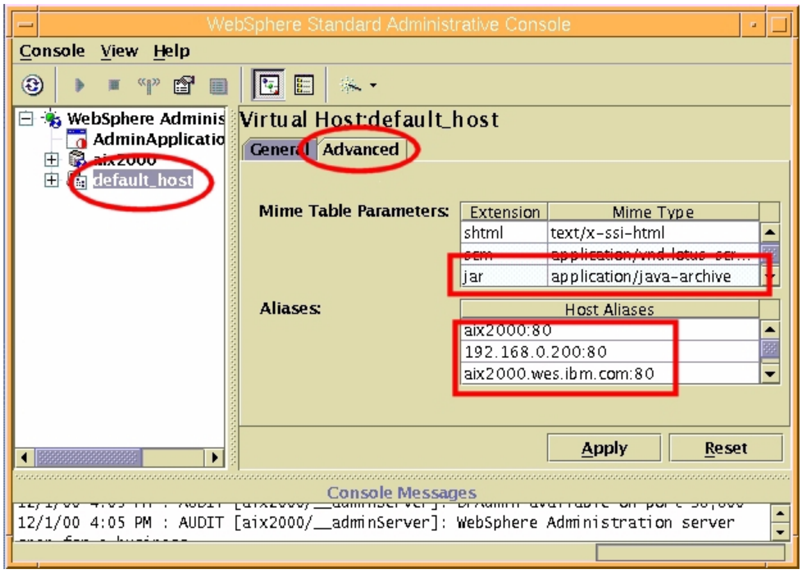
   This will take several minutes to come up.

```
aix2000[/]#cd /usr/WebSphere/AppServer/bin
aix2000[/usr/WebSphere/AppServer/bin]#./startupServer.sh &
[1] 24536
aix2000[/usr/WebSphere/AppServer/bin]#tail -f /usr/WebSphere/AppServer
/logs/tracefile
[00.12.01 15:04:39:902 JST] 980101ae ActiveServerP A Stopped server:
"DMS_AppServer" (pid "19118")
InstantDB - Version 3.13
Copyright (c) 1997-2000 Instant Computer Solutions Ltd.
[00.12.01 15:08:13:903 JST] a2b3c09e AdminServer   A Initializing WebSphere
Administration server
[00.12.01 15:08:13:917 JST] 710c095 DrAdminServer A DrAdmin available
on port 38,152
[00.12.01 15:09:03:932 JST] a2b3c09e AdminServer   A WebSphere
Administration server open for e-business
^C
aix2000[/usr/WebSphere/AppServer/bin]#./adminclient.sh
No remote host or port argument specified. This host and port 900 will
be used.
```

Figure 45. Configuration on WebSphere Application Server panel

2. Expand the tree node **WebSphere Administrative Domain**.
3. Click **default_host** (shown in Figure 46.)
   a. On the right pane, click the **Advanced** tab.
   b. Add alias `<your fully qualified server>:80`.

      If you want, also add alias `<IP address of server>:80` and `<hostname>:80`.
   c. Add an entry to the MIME table parameters:
      - **Extention**: jar
      - **MIME Type**: java/application-archive
   d. Click **Apply**.



Mime Table Parameters:

Extension	Mime Type
shtml	text/x-ssi-html
scm	application/vnd.lotus-scr...
jar	application/java-archive

Aliases:

Host Aliases
aix2000:80
192.168.0.200:80
aix2000.wes.ibm.com:80

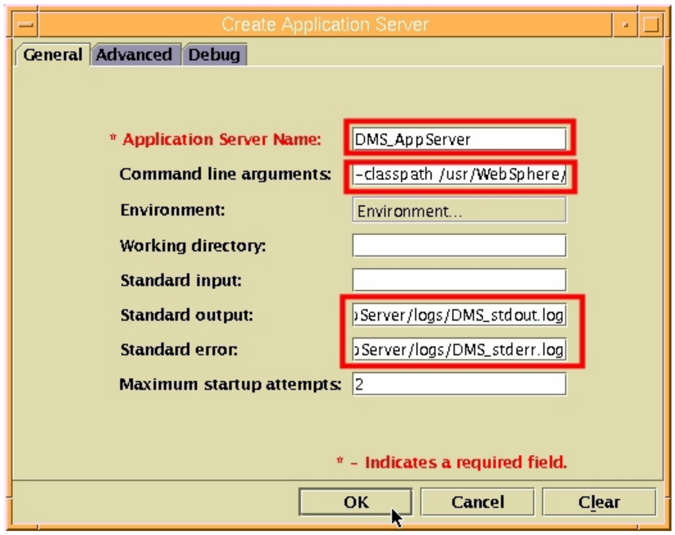
```
12/1/00 4:05 PM : AUDIT [aix2000/__adminServer]: WebSphere Administration server
```

Figure 46. Advanced setting for default_host

The MIME type setting for jar files is required so that Netscape Navigator browsers can correctly handle downloading the Device Manager console files.

4. Create the application server.

   a. Right-click **<your hostname>**. Select **Create -> Application Server** from the list.



* Application Server Name: DMS_AppServer

Command line arguments: -classpath /usr/WebSphere/

Environment: Environment...

Working directory:

Standard input:

Standard output: Server/logs/DMS_stdout.log

Standard error: Server/logs/DMS_stderr.log

Maximum startup attempts: 2

* - Indicates a required field.

Figure 47. Create Application Server window

b. Specify the following attributes for the Create Application Server window.

- **name**: `DMS_AppServer`
- **command line arguments**:

```
-classpath
/usr/WebSphere/AppServer/hosts/default_host/dmserver/servlets:
/usr/lpp/TivDMS/doc/dmapi.jar:
/usr/lpp/TivDMS/doc/log.jar:
/usr/lpp/TivDMS/doc/logging.jar:
/usr/lpp/TivDMS/doc/util.jar:
/usr/lpp/TivDMS/doc/iTk_core.jar:
/usr/lpp/TivDMS/doc/classes12_01.zip:
/usr/lpp/TivDMS/Class/baseplugin.jar:
/usr/lpp/TivDMS/Class/palmplugin.jar:
/usr/lpp/TivDMS/Class/iadplugin.jar:
/usr/lpp/TivDMS/Class/aero8000plugin.jar:
/usr/lpp/TivDMS/Class/win32plugin.jar:
```

If you use the Sun Solaris platform, specify the following command line arguments:

```
-classpath
/opt/WebSphere/AppServer/hosts/default_host/dmserver/servlets:
/opt/TivDMS/doc/dmapi.jar:
/opt/TivDMS/doc/log.jar:
```

```
/opt/TivDMS/doc/logging.jar:
/opt/TivDMS/doc/util.jar:
/opt/TivDMS/doc/iTk_core.jar:
/opt/TivDMS/doc/classes12_01.zip:
/opt/TivDMS/Class/baseplugin.jar:
/opt/TivDMS/Class/palmplugin.jar:
/opt/TivDMS/Class/iadplugin.jar:
/opt/TivDMS/Class/aero8000plugin.jar:
/opt/TivDMS/Class/winceplugin.jar
```

In addition, we recommend that you change the log file names. If you only specify the name of the log files, they are placed in `/usr/WebSphere/AppServer/bin`. If you want to place them in another path, specify the whole path as `/usr/WebSphere/AppServer/logs/DMS_stdout.log`

c. Click **OK**.

5. Create the DMS servlet engine using the following instructions:

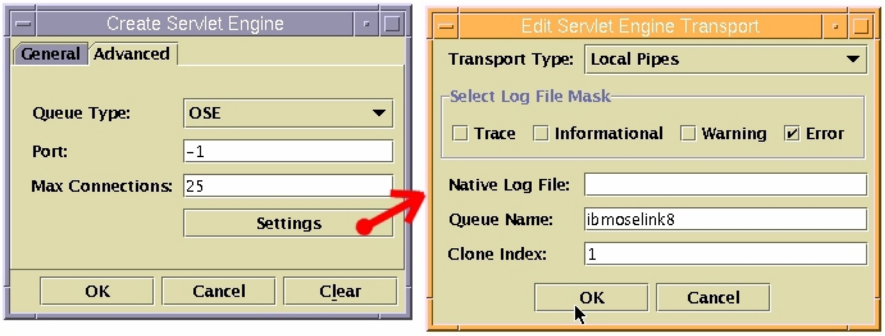
   a. Right-click **DMS_AppServer**. Select **Create -> Servlet Engine** from the list.

   b. In the **General** tab, type `DMS_ServletEngine` for the name of the servlet engine.

   c. Click the **Advanced** tab.

   d. Click **Settings**. Add `ibmoselink8` for the **Queue Name**.

   e. Click **OK**.



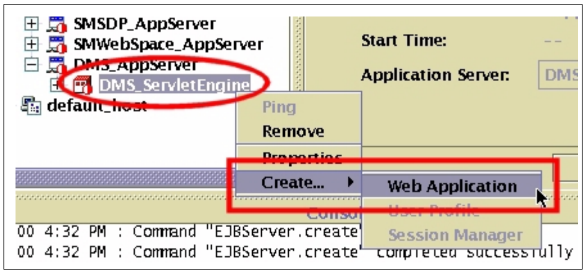
Queue Type: OSE
Port: -1
Max Connections: 25

Transport Type: Local Pipes
Select Log File Mask
□ Trace □ Informational □ Warning ☑ Error
Native Log File:
Queue Name: ibmoselink8
Clone Index: 1

Figure 48. Advanced setting for Servlet Engine

6. Create the Web application.

   a. Right-click **DMS_ServletEngine** in the left pane. Select **Create -> Web Application** from the list.



```
00 4:32 PM : Command "EJBServer.create'
00 4:32 PM : Command "EJBServer.create'  completed successfully
```

Figure 49. Creating a Web application

b. In the **General** tab, specify the following attributes:

- **name**: dmserver
- **web path**: /dmserver

c. Click **OK**.

7. Create the File Serving Enabler, using the following instructions:

a. On the menu bar, select **Console->Task**.

b. Select **Add a servlet**.

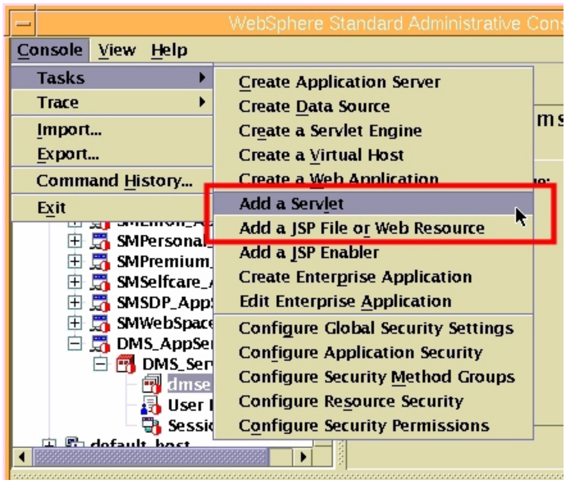

WebSphere Standard Administrative Cons
Console
View
Help
Tasks
Trace
Import...
Export...
Command History...
Exit
Create Application Server
Create Data Source
Create a Servlet Engine
Create a Virtual Host
Create a Web Application
Add a Servlet
Add a JSP File or Web Resource
Add a JSP Enabler
Create Enterprise Application
Edit Enterprise Application
Configure Global Security Settings
Configure Application Security
Configure Security Method Groups
Configure Resource Security
Configure Security Permissions
SMPersonal
SMPremium
SMSelfcare_
SMSDP_App
SMWebSpace
DMS_AppSer
DMS_Ser
dmse
User
Sessi
default_host
ms

Figure 50. Selecting Add a Servlet

c. In the first window, select **No** and click **Next**.

d. Expand all the nodes and select **dmserver**. Click **Next**.

e. Select **Create File-Serving Servlet**.

f. Click **Finish**.

8. Create Device Manager servlet.

   a. Right-click **dmserver** in the left pane. Select **Create -> Servlet**. Specify the following attributes for the **General** tab.

      - **name**: DeviceManagementServerServlet
      - **class**: com.tivoli.dms.dmserver.DeviceManagementServerServlet
      - **Web path**: /dmserver/DeviceManagementServerServlet

   b. Click the **Advanced** tab.

   c. Enter the following Init parameters:

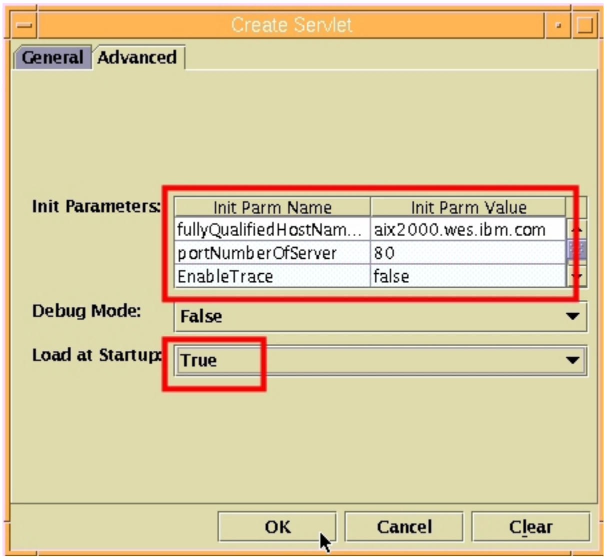
      - **fullyQualifiedHostNameOfServer**: for example, aix2000.wes.ibm.com
      - **portNumberOfServer**: 80
      - **EnableTrace**: false
      - **EnableAPITrace**: false
      - **LogDebugToPrintStream**: false

   d. In an Everyplace Suite environment, using the Everyplace Authentication Server, set the following Init parameter in addition to the previous five parameters. The value for this parameter will differ according to the setting of Authentication Server. For more detailed information on this parameter setting, refer to *Tivoli Internet and Personalized Services Manager: Planning and Installation*.

      authProxyDmsUrl: http://auth.proxy/DMS/

   e. For **Load at Startup,** select **True**.

   f. Click **OK**.



Init Parameters:

Init Parm Name	Init Parm Value
fullyQualifiedHostNam...	aix2000.wes.ibm.com
portNumberOfServer	80
EnableTrace	false

Debug Mode: False

Load at Startup: True

Figure 51. Advanced setting for DeviceManagementServerServlet

9. Create the Palm Servlet (optional).

   a. Right-click **dmserver**. Select **Create -> Servlet**. Specify the following attributes for the **General** tab.

      - **name**: PalmServlet
      - **class**: com.tivoli.dms.plugin.palm.PalmServlet
      - **web path**: /dmserver/PalmServlet

   b. Click the **Advanced** tab.

   c. For **Load at Startup,** select **True**.

   d. Click **OK**.

10. Create Aero 8000 Servlet (optional).

   a. Right-click **dmserver**.

   b. Select **Create -> Servlet**. Specify the following attributes for the **General** tab.

      - **name**: Aero8000Servlet
      - **class**: com.tivoli.dms.plugin.win.wince.hpcpro.aero8000.Aero8000Servlet
      - **web path**: /dmserver/Aero8000Servlet

   c. Click the **Advanced** tab.

   d. For **Load at Startup**, select **True**.

   e. Click **OK**.

11. Create a Windows CE Servlet (optional).

   a. Right-click **dmserver**. Click **Create -> Servlet**. Specify the following attributes for the **General** tab:

      - **name**: WinceServlet
      - **class**: com.tivoli.dms.plugin.win.wince.WinceServlet
      - **web path**: /dmserver/WinceServlet

   b. Click the **Advanced** tab.

   c. For **Load at Startup**, select **True**.

   d. Click **OK**.

12. Create a PvC Client Stack Servlet (optional).

   a. Right-click **dmserver** in the left pane.

   b. Click **Create -> Servlet**. Specify the following attributes for the **General** tab:

      - **name**: IadServlet
      - **class**: com.tivoli.dms.plugin.pvc.iad.IadServlet
      - **web path**: /dmserver/IadServlet

   c. Click the **Advanced** tab.

   d. For **Load at Startup**, select **True**.

   e. Click **OK**.

   Now you can start the Device Manager server.

13. Start the Web server by issuing the following command:

   ```
   /usr/HTTPServer/bin/apachectl start
   ```

14. In the WebSphere Administration GUI, select the node **DMS_AppServer** in the left pane.

15. On the menu bar, click the **play** button (the triangle on its side.) A message stating that the application server was successfully started is displayed.

The status window should say “DMS_AppServer started successfully”, and there should be no errors in the log files that are stored in the /usr/WebSphere/AppServer/bin/stdout.txt and stderr.txt paths by default.

To access any of the DMS HTML including the Device Manager console installation program, WebSphere must be running and the dmserver Web Application must be active.

The next section describes the whole sequence of starting a DM server.

Turning on the trace

Tracing is off by default, but can be turned on by changing the following two parameters in the Advanced tab of DeviceManagementServerServlet:

- **EnableTrace**: true
- **LogDebugToPrintStream**: true

If tracing is on, logs are stored in DMS_stdout.log and DMS_stderr.log. The size of the log files can increase to a great extent. Check them often.

3.4.3 Starting the Device Manager server

Now that all the configuration tasks have been performed, you can start the Device Manager server.

3.4.3.1 Checklist

Before starting the Device Manager server, check the following factors. This list is useful also for troubleshooting:

- TPSM database
  - Are the tables are properly created?
  - Are all the required device classes and job classes registered?
  - Is the database and the listener properly started?
- Device Manager setting
  - Is the configuration on WAS correct?
  - Are properties files properly set? Check the following files. Section 3.4.5, “Properties files” on page 99, offers detailed information on these properties files.
    - Transaction.properties
    - SubscriptionMgr.properties
    - authentication.properties
    - tsmauthentication.properties

- Subscription Manager server

  Is the Subscription Manager server properly up and running before starting up the Device Manager server?

3.4.3.2 Starting the server

Before you start the Device Manager server, start the following servers in the order specified. These instructions assume that Device Manager and Subscription Manager are on the same machine.

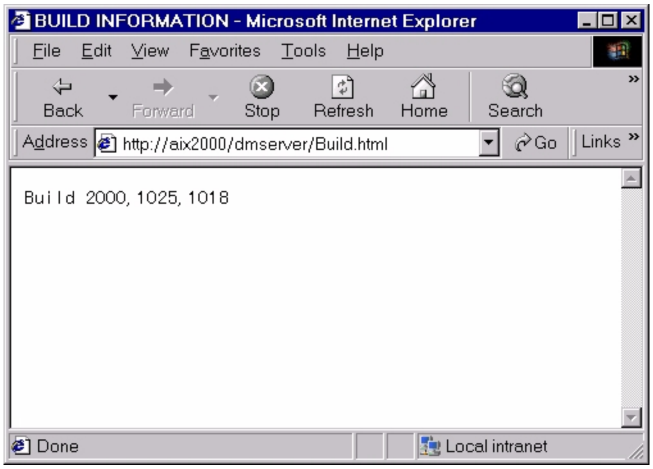
1. On the TPSM database server, start the database instance and the listener.
2. On the TPSM server, start the HTTP server daemon.
3. Start WebSphere Application Server and its administration client on the TPSM server.
4. Start Subscription Manager services from the WAS administration client window.

   Next, you can start the Device Manager server, using the following instructions:
5. In the WAS administration client window, select the **DMS_AppServer** node in the left pane.
6. On the menu bar, click the **play** button (the triangle on its side.) A message stating that the application was successfully started is displayed.

You can confirm whether the Device Manager server has successfully started by accessing the URL:

```
http://dmserver_hostname/dmserver/Build.html
```

If you see the page shown in Figure 52 on page 98, this means FileServingEnabler of the Device Manager server started successfully.


BUILD INFORMATION - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Search
Address
http://aix2000/dmserver/Build.html
Go
Links
Done
Local intranet

Build 2000, 1025, 1018

Figure 52. Build number indicating the server started successfully

You can see start up logs in the file DMS_stdout.log and DMS_stderr.log. The paths and the file names are system dependent because these names are specified as the configuration parameters of DeviceManagementServerServlet. You can confirm the path and the name of these log files in the WAS administration client console.

3.4.3.3 Stopping the Device Manager server

You can stop the Device Manager server by using the following instructions:

1. Stop the Device Manager server.

   To stop the Device Manager server, select the **DMS_AppServer** application server in WAS administration client's left pane. Then click the **Stop** button (the black square button in the window.)

   If you are going to shut down the system, use the following instructions in addition to stopping the Device Manager server.

2. Stop WebSphere Application Server.

   If other Subscription Manager servers are on the same machine, stop all the application servers on the system. Then, select the icon hostname in the left pane of the WAS administration client and click the **Stop** button (the black square in the window.) This action stops all the WAS daemons and closes the administration client window. After the window is closed, confirm that there are no Java processes left on the system by issuing the following command:

   ```
   ps -ef grep java
   ```

3. Stop the HTTP server and shut down the system.

3.4.4 Configuration differences

This section describes the configuration differences between Device Manager in an Everyplace Suite environment and in a non-Everyplace Suite environment.

From the viewpoint of tasks you must perform, if you deploy Device Manager in a non-Everyplace Suite environment, you should configure the WAS for Device Manager server manually. In Everyplace Suite, Everyplace Suite installer performs the installation and almost all configuration tasks on WAS automatically.

However, settings in WebSphere Application Server have only one difference between the two installations. It is the parameter of DeviceManagementServerServlet, named `AuthProxyDmsUrl`. In an Everyplace Suite environment, this parameter is the only parameter to be set manually, and required so that Authentication Server can be properly passed. In a non-Everyplace Suite environment, the `AuthProxyDmsUrl` parameter is not required.

3.4.5 Properties files

Device Manager controls its authentication configuration connection to the Device Manager database by using some configuration files. According to your system requirements, edit these files before you start up the Device Manager server. Table 6 contains the list of the key properties files.

Table 6. Properties files

File name	**Description**
Transaction.properties	Contains the information to connect to the Device Manager database server
SubscriptionMgr.properties	Contains the class name of the Subscription Manager, which is used as the enrollment application
authentication.properties	Contains the name of the class that is used to authenticate the DM console administrator
tsmauthentication.properties	Contains the name of the profile that is used as the DM console administrator profile

You can manually edit these files if you want to change the database server or the administrator’s profile or for other reasons.

3.4.5.1 The Transaction.properties file

This file is in the path `/usr/lpp/TivDMS/Class` (AIX) or `/opt/TivDMS/Class` (Solaris) and contains critical database connectivity information. After the Device Manager installation, if you want to change the database server, or in case of trouble, you should confirm or update this file.

If you are using Oracle8i as the TPSM database, the file is in a format similar to the format shown in Figure 53. The items you need to check appear in bold italic.

```
JDBC.dbDriver=oracle.jdbc.driver.OracleDriver
JDBC.dbConnect=jdbc:oracle:thin:@host_name:port_number:database_name
JDBC.dbUser=user_ID
JDBC.dbPassword=password
MaxDBConnections=200
MinDBConnections=1
EnableRMI=false
enabletrace=false
Thread_Control=true
Queue_Length=-1
```

Figure 53. The Transaction.properties file using Oralcle8i as the TPSM database

If you are using DB2 Universal Database as the TPSM database, the file is in a format similar to the format shown in Figure 54. The items you need to check appear in bold italic.

```
JDBC.dbDriver=COM.ibm.db2.jdbc.net.DB2Driver
JDBC.dbConnect=jdbc:db2://host_name:port_number/database_name
JDBC.dbUser=user_ID
JDBC.dbPassword=password
MaxDBConnections=200
MinDBConnections=1
EnableRMI=false
enabletrace=false
Thread_Control=true
Queue_Length=-1
```

Figure 54. The Transaction.properties file using DB2 UDB as the TPSM database

To improve Device Manager performance, set the database connections higher in Transaction.properties. By default, MinDBConnections is set to 1, and MaxDBConnections is set to 200.

It is more efficient to open a larger number of database connections at startup rather than later, during run time. Later, opening database connections slows server capabilities to process requests, which affects device clients, and therefore, subscribers.

For a thousand concurrent device clients, 70 to 100 connections being used is typical. Use this as a guideline to set the MinDBConnections. MaxDBConnections should be larger, but not more that the system can realistically handle. The recommended maximum is 256 or 512.

DB2 specific settings

If you are using DB2 Universal Database as the relational database, increase the Maximum Locks DB2 configuration parameter from 10 (which is the default value) to 20.

3.4.5.2 SubscriptionMgr.properties

This file contains the information about Subscription Manager that is used as the device enrollment application by Device Manager. This file contains the class name of the Subscription Manager and some additional information as shown in Figure 55.

```
#---------------------------------------------------------------------
#   5698-PSM
# (C) Copyright Tivoli Systems Inc., an IBM Company 2000. All rights reserved.
#
# The source code for this program is not published or otherwise
# divested of its trade secrets, irrespective of what has been
# deposited with the U.S. Copyright Office.
#---------------------------------------------------------------------
CLASS_NAME=com.tivoli.dms.dmapi.tsm.TSMSubscriptionMgr
USER_UPPERCASE=true
REALM_UPPERCASE=true
OFFERING_UPPERCASE=false
```

Figure 55. SubscriptoinMgr.properties file

The class name of the Subscription Manager in TPSM is com.tivoli.dms.dmapi.tsm.TSMSubscriptionMgr. This is the default setting. However, you can use any other subscription application, providing that the following functions are included:

- The subscription management product must provide an enrollment application that registers new devices in the Device Manager database upon enrollment.
- The enrollment application must ensure that each registered device has only one owner, the subscriber, associated with it.

- The enrollment application must know how to interact with supported devices to obtain the necessary enrollment information.
- The Subscription Manager must provide a Java exit that validates realm, deal, and user information, lists valid realms and deals, and handles errors. The exit shipped with Device Manager is called TSMSubscriptionMgr.java, and it is customized for Tivoli Personalized Services Manager. The SubscriptionMgr.properties file contains a pointer to the class name for the exit.

TPSM does not have a toolkit to develop the subscription application that satisfies the functions described above.

If you want to use a subscription management product other than TPSM, update this file to point to the class name of the product.

Both the TPSM Subscription Manager and Device Manager use the same database. They use their own tables, which have no relation to each other, but they store duplicate data in both tables. Because the database is case-sensitive, Device Manager must store and retrieve data in the same case that Subscription Manager is using. To settle this, the SubscriptionMgr.properties file has three keyword/value pairs:

- USER_UPPERCASE (default: *true*)
- REALM_UPPERCASE (default: *true*)
- OFFERING_UPPERCASE (default: *false*)

Set these values to *true* or *false*, depending on how the Subscription Manager stores data in these fields. The value *true* is for uppercase data, and *false* is for lowercase data.

3.4.5.3 Authentication.properties

This file contains the name of the class that is used to authenticate the Device Manager console administrator. This is located at `/opt/TivDMS/doc` on Solaris or at `/usr/lpp/TivDMS/doc` on AIX.

During installation, you can choose the class from two options:

- Tivoli Personalized Services Manager authentication
- Other Authentication

Use the Tivoli Personalized Services Manager authentication to give access to a predefined group of authorized Tivoli Personalized Services Manager administrators. This is the default authentication. If Tivoli Personalized Services Manager authentication is specified during installation, the authentication.properties file is updated with the class com.tivoli.dms.console.security.tsm.TSMAuthentication.

The class used for TPSM authentication is located in the Device Manager classpath. This class is specified as a keyword = value statement in the authentication.properties file, as shown in Figure 56.

```
#----------------------------------------------------------------------
#   5698-PSM
# (C) Copyright Tivoli Systems Inc., an IBM Company 2000.  All rights reserved.
#
# The source code for this program is not published or otherwise
# divested of its trade secrets, irrespective of what has been
# deposited with the U.S. Copyright Office.
#----------------------------------------------------------------------
authClass = com.tivoli.dms.console.security.tsm.TSMAuthentication
```

Figure 56. The authentication.properties file

Use other authentication if you or an integrator wants to write your own Java class for some alternative authentication method, which Device Manager then instantiates. If you specify this authentication at install time, the installation program asks for the name of the authentication class to be instantiated. You must then supply the name of the Java class you created. The installation program adds this name to the authentication.properties file, for example:

```
authClass = com.tivoli.dms.console.security.MyAuthentication
```

It is up to you or the integrator to install this class on the administration client computers that use the console – either by copying the class to each computer or by adding it to the console.jar file that is downloaded to administration clients when you install the console.

authClass for test environments

Device Manager provides an authentication class for a test environment, named com.tivoli.dms.console.security.DefaultAuthentication.

It allows successful login by only two users:

- Administrator/password
- a/pw

To switch to this, choose one of the following options:

- Choose **Other** during installation, and enter the class `com.tivoli.dms.console.security.DefaultAuthentication`
- Edit your authentication.properties to read: authClass = `com.tivoli.dms.console.security.DefaultAuthentication`

This authentication allows users to access all the data in the Device Manager database tables. Use this only in the test phase or for troubleshooting. This authentication class is developed just for the test environment, in spite of its name "DefaultAuthentication".

The default authentication class is com.tivoli.dms.console.security.tsm.TSMAuthentication, which is called as TPSM authentication.

For more information about writing and implementing your own authentication class, see *Device Manager: Developer's Guide*, and *Device Manager: Administration*, which are provided as DM online documents.

3.4.5.4 tsmauthentication.properties

This file specifies the profile of the Device Manager console administrator. This is located at `/opt/TivDMS/doc` on Solaris or at `/usr/lpp/TivDMS/doc` on AIX. If you select TPSM authentication during installation, you are also asked for the name of a Tivoli Personalized Services Manager profile identifying a group of authorized administrators' user IDs. The default TPSM profile name is *Device Admin*. This profile sets no limit to access the Device Manager database, so the administrator can access any of the device management data. The file is shown in Figure 57.

```
#-------------------------------------------------------------------
#   5698-PSM
# (C) Copyright Tivoli Systems Inc., an IBM Company 2000.  All rights reserved.
#
# The source code for this program is not published or otherwise
# divested of its trade secrets, irrespective of what has been
# deposited with the U.S. Copyright Office.
#-------------------------------------------------------------------
profile = Device Admin
```

Figure 57. The tsmauthentication.properties file

If a Device Admin profile does not already exist in Tivoli Personalized Services Manager, you can add one using its Director Tool. For information on how to add a new profile, see “Access Control Configuration” in *Tivoli Internet and Personalized Services Manager: Director Tool*. It describes how to set up security profiles for system administrators.

If you want to perform any access control to the database, create a profile with Director tool and specify the profile name in this file. After you update the file, restart the Device Manager application server to activate the change.

3.5 References

These are links to TPSM-related product documentation needed during installation:

- Device Manager online documents

  Device Manager provides following online documents:

  - *Tivoli Personalized Services Manager Device Manager: Planning and Installation*
  - *Tivoli Personalized Services Manager Device Manager: Administration*
  - *Tivoli Personalized Services Manager Device Manager: Developer’s Guide*
  - *Tivoli Personalized Services Manager Device Manager: Device Plug-in Notes*

  You can access them by:

  - The Help menu on the Device Manager console window
  - The Internet at: `http://hostname/dmserver/web/en/index.htm`

- *Tivoli Internet and Personalized Services Manager: Programmer’s Guide*

- IBM Redbook: *Introducing Tivoli Personalized Services Manager 1.1*, SG24-6031
- IBM AIX HACMP
  - HACMP V4.4 publications Online documents

    `http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/hacmp_index.html#V44`

    The manual *HACMP for AIX 4.4 Planning Guide*, SC23-4277, is useful for planning the HACMP system, which is located on the previously mentioned Web site.
- IBM HTTP Server
  - IBM Software: IBM HTTP Server Web site:

    `http://www.ibm.com/software/webservers/httpservers/`
  - IBM Redbook: *IBM HTTP Server Powered by Apache on RS/6000*, SG24-5132
- IBM WebSphere Application Server
  - IBM Software: WebSphere Application Server product Web site:

    `http://www.ibm.com/software/webservers/appserv/`

    The home for IBM WebSphere Application Server, see this Web site for multiplatform information about the product.
  - IBM Redbook: *WebSphere Application Servers: Standard and Advanced Editions*, SG24-5460

    Although this redbook focuses on the AIX and Windows NT platforms and older version of WAS, it contains good information about the WebSphere Application Server product in general, such as planning for and troubleshooting WebSphere Application Server.
- Database systems
  - IBM Redbook: *Database Performance on AIX in DB2 UDB and Oracle Environments*, SG24-5511
- IBM Network Dispatcher
  - IBM Software:

    `http://www.ibm.com/software/webservers/edgeserver`

    From the Web site, you can locate *User's guide (Version 3.0 for Multiplatforms)*, GC31-8496
  - IBM Redbook: *IBM WebSphere Performance Pack: Load Balancing·with IBM SecureWay Network Dispatcher*, SG24-5858

The current product name is IBM Network Dispatcher, which is a feature of WebSphere Edge Server.

- Java resources
  - Sun Microsystems Java Technology Web site:

    `http://java.sun.com`

    Sun home page for Java technology, this Web site is a valuable resource for Java news, specifications, products, APIs, and documentation.
  - Sun JDK Web site:

    `http://www.sun.com/software/solaris/java/download.html`
  - JDBC driver (DB2 UDB)

    This is included in the DB2 product.
  - JDBC driver (Oracle. Registration is free and is required).

    `http://technet.oracle.com/software/tech/java/sqlj_jdbc/software_index.htm`
  - General information about JDBC Web site:

    `http://java.sun.com/products/jdbc/`

Chapter 4. Administration

This chapter describes how to perform administration tasks using Device Manager. Most administration tasks are performed from a Microsoft Windows computer by using the Device Manager console.

4.1 Overview

There are two types of administration tasks to provide device management support for service subscribers. Most administration tasks are performed from Device Manager console, and some are performed from the Device Manager server.

Device Manager server tasks are performed by keying a command from a command line interface of the Device Manager server. You can perform the following tasks:

- Manage device classes
- Manage job types and classes
- Change administration authentication
- Manage large numbers of jobs or devices

Detailed explanations are provided in 4.2, “Device Manager server tasks” on page 110.

Administrators should also monitor important log files on the Device Manager server for error and informational messages.

Device Manager console tasks are management tasks that administrators can perform such as, manage devices, device classes, jobs, and software in your network and display Device Manager server information. Typical console tasks include:

- Managing devices
- Managing jobs
- Managing software

Detailed explanations for each task can be found in 4.3, “Device Manager console tasks” on page 125.

You can install and start the Device Manager console from the Device Manager server by locating the following Web site:

```
http://server_name/dmserver/DMconsole.html
```

For more information about how to install and customize the Device Manager console, refer to 4.3.1, “Device Manager console installation” on page 125.

4.2 Device Manager server tasks

In this section, we describe Device Manager server tasks that need to be performed from a Device Manager server. The following section summarizes directory structure differences between operating systems and introduce several important operation commands.

To install and customize the Device Manager server, refer to Chapter 3, “Installation and configuration” on page 55.

4.2.1 Directories and files

You need to know directory structures and some files that are important to manage the Device Manager server as an administrator.

The Device Manager server runs on the IBM AIX system or the Sun Solaris system, and their directory structures are different from each other. From now on, when we mention directory and file names, we are referring to only IBM AIX system names. If your system is a Sun Solaris, replace the names according to the information illustrated in Figure 58, which shows the Device Manager server related directories.

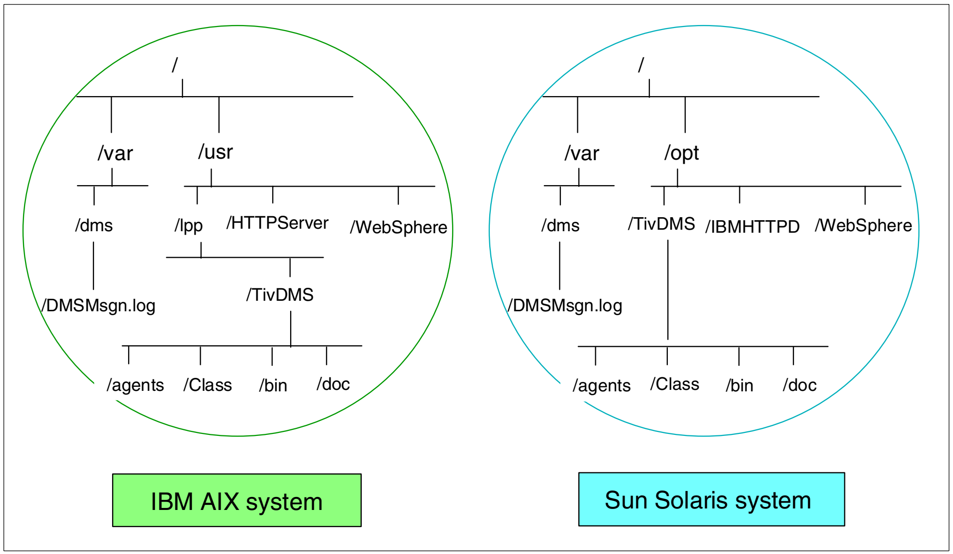


Figure 58. DMS related directories of the IBM AIX and Sun Solaris systems

There are important directories and files that you need to know how to manage. Table 7 summarizes the directories and files used by the Device Manager server.

Table 7. DMS directories and files

Directory	File or files	Description
/usr/lpp/TivDMS/ Class	In this directory, there are some jar files using by IBM WebSphere Application Server, and some property files.	
	Transaction.properties	A properties file for the Device Manager database, includes: server address and db user information
	SubscriptionMgr.properties	A properties file of the Subscription Manager
	<plug-in name>.jar	Jar files for each device plug-ins
/usr/lpp/TivDMS/ bin	In this directory, there are template files, DDL files and shell script files for administrator	
	<plug-in name>.template	Template files for each device plug-ins
	createDMSxxxx.sql	The DDL files to create Device Manager database
	dms_addplugins.sh	A shell script to install device classes supplied with Device Manager server 1.1
	devclasscfg.sh	A shell script to handle device classes
	jobclasscfg.sh	A shell script to handle job classes
	delcompjobs.sh	A shell script to delete completed jobs
	DMSUtil.sh	a shell script to manage the large number jobs

Directory	File or files	Description
/usr/lpp/TivDMS/doc	In this directory, there are some property files and jdbc driver, and many documentation files	
	authentication.properties	A properties file for authentication of the Device Manager console.
	classes12_01.zip	A jdbc driver
	lookup.properties	A properties file used during Device Manager console installation
	tsmauthentication.properties	A properties file for the TPSM authentication method.
/usr/lpp/TivDMS/agents	This directory has some directories, each of them have device name. There are client software under these directories	
	<device name>/	Client software for each plug-in. Refer to 7.1, “Device agent installation” on page 193
/var/dms	DMSMsgn.log	A Device Manager server log file
/usr/WebSphere/AppServer/logs *	DMS_stdout.log	A informational message file
	DMS_stderr.log	A error message file
	* **Note**: You can change these file names, refer to 4.2.6, “Device Manager server log files” on page 124	

4.2.2 Managing a device class

A device class is a group of devices with similar characteristics and functions. A new device class must be installed and configured on the Device Manager server before any devices of that class are added to, or managed by, Device Manager.

The installation and configuration tasks for device classes are run from a command line script named `devclasscfg.sh` on the Device Manager server.

The tasks for managing a device class include:

- Installing a new device class
- Deleting a device class
- Modifying a device class
- Listing device classes

The characteristics and functions of a device class are stored as properties in the DEVICE_CLASS data table in the Device Manager database.

The syntax of the `devclasscfg.sh` command is shown in Figure 59.

```
devclasscfg.sh -add DeviceClassName -class JavaClass [-template TemplateFile]\
                     [-enroll Enrollment URL] [-description "Description"]

devclasscfg.sh -delete DeviceClassName

devclasscfg.sh -modify DeviceClassName [-name NewDeviceClassName] [-class JavaClass]\
      [-template TemplateFile] [-enroll Enrollment URL] [-description "Description"]

devclasscfg.sh -list [DeviceClassName]
```

Figure 59. Syntax for devclasscfg.sh command

For more information about supported parameters with `devclasscfg.sh`, refer to the “Parameters for devclasscfg.sh” in the online manual *Tivoli Personalized Services Manager Device Manager: Administration*.

4.2.2.1 Installing a new device class

When you install and configure a new device class, you must follow these steps:

1. Verify that the Device Manager database is started and working properly.
2. Copy the Java class files that implement the device class to the application server classpath on the Device Manager server or servers.
3. Use the `devclasscfg.sh -add DeviceClassName` command:

   ```
   devclasscfg.sh -add DeviceClassName -class JavaClass -enroll
   EnrollmentURL -template TemplateFile -description Description
   ```

4. Restart the device management server(s) servlet to create an instance of the device class.

The `devclasscfg.sh -add DeviceClassName` command creates a DEVICE_CLASS table entry in the Device Manager database used by all of the Device Manager servers.

The `dms_addplugin.sh` command, used during Device Manager server installation, also calls the `devclasscfg.sh` command.

For more information about the `dms_addplugin.sh` command, refer to 3.4.1, “Registering the device plug-in classes” on page 81.

4.2.2.2 Deleting a device class

You can delete a device class using the `devclasscfg.sh` command. When you delete a device class from the database, the following related information is deleted as well:

- All device class configuration parameters for the device class
- All job classes for the device class
- The software objects for the device class
- All job entries for the device class

Deleting a device class only updates the Device Manager database, that is, Java class files for the deleted device class still remain on the server. You must delete the Java class files from each Device Manager server manually.

You cannot delete a device class if physical devices of that device class exist in the DEVICES table in the Device Manager database.

4.2.2.3 Modifying device class properties

You can modify device class properties using the `devclasscfg.sh` command. The following properties can be changed:

- Device class name
- Description
- Java class name
- Enrollment URL
- Configuration template file

The description and the enrollment URL can also be modified from the Device Manager console.

Modifying a device class with the `devclasscfg.sh` command only updates the information in the Device Manager database.

The `devclasscfg.sh` command does not change the Java class files, that is, you must change Java class files on each Device Manager server manually.

4.2.2.4 Listing device classes

To list (display information about) a device class, use the following command from the Device Manager server:

```
devclasscfg.sh -list DeviceClassName .
```

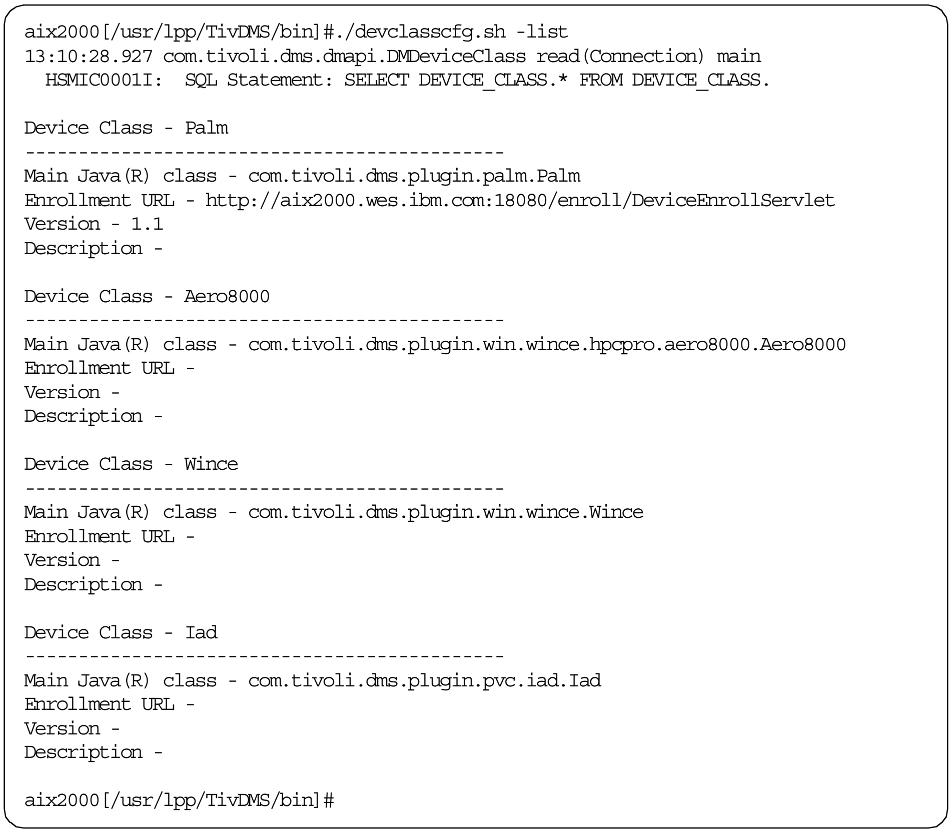
The DeviceClassName for the -list parameter is optional.

If DeviceClassName is specified, the device class fields are displayed, and the configuration template file fields for that device class are displayed.

You have the same output when you use the SQL statement:

```
SQL> select * from DEVICE_CLASS where DEVICE_CLASS_NAME = 'Palm';
```

If DeviceClassName is omitted, all device classes are displayed, but no configuration template file fields are displayed as shown in Figure 60.



```
aix2000[/usr/lpp/TivDMS/bin]#./devclasscfg.sh -list
13:10:28.927 com.tivoli.dms.dmapi.DMDeviceClass read(Connection) main
  HSMIC0001I:  SQL Statement: SELECT DEVICE_CLASS.* FROM DEVICE_CLASS.

Device Class - Palm
---------------------------------------------
Main Java(R) class - com.tivoli.dms.plugin.palm.Palm
Enrollment URL - http://aix2000.wes.ibm.com:18080/enroll/DeviceEnrollServlet
Version - 1.1
Description -

Device Class - Aero8000
---------------------------------------------
Main Java(R) class - com.tivoli.dms.plugin.win.wince.hpcpro.aero8000.Aero8000
Enrollment URL -
Version -
Description -

Device Class - Wince
---------------------------------------------
Main Java(R) class - com.tivoli.dms.plugin.win.wince.Wince
Enrollment URL -
Version -
Description -

Device Class - Iad
---------------------------------------------
Main Java(R) class - com.tivoli.dms.plugin.pvc.iad.Iad
Enrollment URL -
Version -
Description -

aix2000[/usr/lpp/TivDMS/bin]#
```

Figure 60. Device class listing example

4.2.3 Managing job types and classes

Multiple job types can be defined for each device class. For example, there is a job type for updating the configuration of a device class and another job type for distributing software to a device class.

For each job type, there is a corresponding device-specific job class to the job type. For example, Device Manager provides a job class for software distribution to Palm devices and another job class for software distribution to Internet appliances.

Remember that even though these two job classes have the same role of software distribution, the device-specific characteristics of these two tasks require a different software distribution job class for each device class.

A job class is written in Java language. You can see all job classes installed in the Device Manager server using the `jobclasscfg.sh` command. For information about how to use the command, see 4.2.3.4, “Listing job classes” on page 117.

The installation and configuration tasks for job types and classes are run from a command line using the `jobclasscfg.sh` command on the Device Manager server.

The managing job type and class tasks include:

- Installing a new job class
- Deleting a job class
- Modifying a job type, the device class, and Java class
- Listing job classes

After a device class has been installed and configured on the Device Manager server, the job class written to implement the job types for a device class must be installed and configured on the server. The job class must be configured before an administrator can submit a job of that job type.

For each job class, the job type, associated device class, and Java class are stored in the JOB_CLASS_TABLE database on the Device Manager server.

The `jobclasscfg.sh` command syntax is shown in Figure 61.

```
jobclasscfg -add -type JobType -deviceclass DeviceClassName -javaclass JavaClassName

jobclasscfg -delete -type JobType -deviceclass DeviceClassName

jobclasscfg -modify -type JobType -deviceclass DeviceClassName [-newtype JobType] \
                    [-newdeviceclass DeviceClassName] [-newjavaclass JavaClassName]

jobclasscfg -list
```

Figure 61. Syntax for jobclasscfg.sh command

4.2.3.1 Installing a job class

When you install and configure a new job class, follow these steps:

1. Verify that the Device Manager database is started and working properly.
2. Copy the Java class files that implement the job class to the application server classpath on the Device Manager server or servers.

3. Issue the `jobclasscfg.sh -add` command.

```
jobclasscfg.sh -add -type JobType -deviceclass DeviceClassName
-javaclass JavaClassName
```

The `jobclasscfg.sh -add` command adds a job class record to the JOB_CLASS_TABLE, which stores the job class definitions in the Device Manager database used by all of the Device Manager servers.

4.2.3.2 Deleting a job class

If you use the `jobclasscfg.sh` command with the `-delete` parameter, a job type and associated device class are deleted from the Device Manager database. You must specify the `-type` and `-deviceclass` parameters to delete a job class.

When a job class is deleted, only the job class record in the Device Manager database is deleted. To delete the Java class files that implement the job type for the device class, you must delete those Java class files from each Device Manager server manually.

When a job class is deleted, any outstanding jobs of that job type generate an error exception when the jobs are attempted.

4.2.3.3 Modifying job class properties

You can modify the following job class properties using the `jobclasscfg.sh` command:

- Job type
- Device class name
- Java class name

Modifying a job class with the `jobclasscfg.sh` command only updates the information in the Device Manager database. Any changes to the Java class files that implement the job type for the device class must be performed at each Device Manager server.

4.2.3.4 Listing job classes

To list (display information about) a job class, use the `jobclasscfg.sh -list` command from the Device Manager server. Figure 62 on page 118 shows a listing of default job classes.

```
aix2000[/usr/lpp/TivDMS/bin]#./jobclasscfg.sh -list

*************** Some lines are deleted *************************

JOB_CLASS_TABLE
---------------
Job Type - SW_DIST
Java class - com.tivoli.dms.plugin.palm.PalmSoftwareDistributionJob
Device Class Name - Palm

Job Type - DEVICE_CFG
Java class - com.tivoli.dms.plugin.palm.PalmDeviceConfigurationJob
Device Class Name - Palm

Job Type - SW_DIST
Java class - com.tivoli.dms.plugin.base.rdmi.RdmiSoftwareDistributionJob
Device Class Name - Aero8000

Job Type - DEVICE_CFG
Java class - com.tivoli.dms.plugin.win.wince.hpcpro.aero8000.Aero8000DeviceConfigurationJob
Device Class Name - Aero8000

Job Type - SW_DIST
Java class - com.tivoli.dms.plugin.base.rdmi.RdmiSoftwareDistributionJob
Device Class Name - Wince

Job Type - DEVICE_CFG
Java class - com.tivoli.dms.plugin.base.rdmi.RdmiDeviceConfigurationJob
Device Class Name - Wince

Job Type - SW_DIST
Java class - com.tivoli.dms.plugin.base.rdmi.RdmiSoftwareDistributionJob
Device Class Name - Iad

Job Type - DEVICE_CFG
Java class - com.tivoli.dms.plugin.base.rdmi.RdmiDeviceConfigurationJob
Device Class Name - Iad

Job Type - RESTPAGE_MGMT
Java class - com.tivoli.dms.plugin.pvc.iad.IadSampleRestPageJob
Device Class Name - Iad

aix2000[/usr/lpp/TivDMS/bin]#
```

Figure 62. Listing of default job classes

4.2.4 Administration authentication

During the installation of the Device Manager server, the type of authentication is defined. It is used to identify Device Manager administrators. You can find which type of authentication is used through the authentication.properties file on the Device Manager server as shown in the following example.

```
authClass = com.tivoli.dms.console.security.tsm.TSMAuthentication
```

There are two types of authentication handling:

- **Tivoli Personalized Services Manager authentication**

  Grants access using a TSMAuthentication class and an existing profile of authorized Tivoli Personalized Services Manager administrators and their user IDs. This is the default authentication.

- **Other authentication**

  Grants access using a Java class for an alternative authentication method written or provided by the service provider.

If you select Tivoli Personalized Services Manager authentication as your method of authenticating Device Manager administrators, the Device Manager installation program asks for the name of a TPSM security profile identifying a group of authentication administrators' user IDs.

The tsmauthentication.properties file contains the security profile name. If your security profile was the default profile, Device Admin, then the file contains the following statement:

```
profile = Device Admin
```

4.2.4.1 Changing to Other authentication

When you want to change your authentication methods to Other authentication from the default Tivoli Personalized Services Manager authentication, perform the following steps:

1. Provide your own authentication class that implements com.tivoli.dms.console.security.AuthenticationInterface. Name this class com.my_company.MyAuthentication, where *my_company* is your company's name.

   For more information about writing and implementing your own authentication class, refer to the online manual *Tivoli Personalized Services Manager Device Manager: Developer's Guide*.

2. Save a backup copy of com.my_company.MyAuthentication. This file will be needed the next time you have to install an updated version of the console. Do not put your backup copy in the DM console directory tree on the administration client computer, because that tree may be overridden during future console updates. We recommend that you keep your backup copy on another computer.

3. Modify the authClass statement in the authentication.properties file on every Device Manager server to read as follows:

   ```
   authClass = com.my_company.MyAuthentication
   ```

4. Copy MyAuthentication.class to the following location on every administration client running the Device Manager console:

   ```
   DMconsole\uid\tivolitivolidms<nn...n>\<your_path>
   ```

   Here, *<nn...n>* is a string that represents a random number created for the current version of the console you installed (you can view this string by looking at the DM console directory path from Windows Explorer after the console is installed). Replace *<your_path>* with the fully qualified path name indicating the location of MyAuthentication.class on the administration client computer (for example, com\my_company). Following is an example of the complete path when the default TPSM authentication method is selected:

   ```
   DMconsole\uid\tivoli_tivolidmsEDA822E5A0E565FB50D6EE75F204B5BF_1_0_0_20
   001025\com\tivoli\dms\console\security\tsm
   ```

The next time the console is started, it will retrieve the new authentication.properties file from the same Device Manager server that was accessed originally to install the console. The console then starts using the new authentication class, MyAuthentication.

If the console fails to communicate with its Device Manager server for any reason, the authentication.properties file will not be retrieved and another attempt will not be made until the next time the console is started.

4.2.4.2 The Device Manager console update

If a new version of the console is installed on the Device Manager server (for example, if you updated Device Manager by installing a PTF), the next time the console is started on each administration client:

- The Device Manager Console Installer will retrieve the new console version.
- All the path names on the administration client that contain the *<nn...n>* random number string will change.

If you use the TPSM authentication, all files will be copied automatically to the administration client.

But, if your authentication type is Other, the MyAuthentication.class file that was copied to the administration client will be lost and unrecoverable. Access to the started Device Manager console will be denied because Device Manager will not be able to find the current MyAuthentication.class file. Therefore, on each administration client, after the started console has retrieved and installed the new console version from the Device Manager server, you must perform the following steps to access the new console:

1. Close the started console.
2. Obtain the backup copy of MyAuthentication.class.
3. Copy MyAuthentication.class to the new location on the administration client running the Device Manager console.

4.2.5 Managing large numbers of jobs or devices

From the command line on any Device Manager server, you can issue the following commands to manage large numbers of jobs or devices efficiently:

- `DMSUtil.sh` command: Manipulates large numbers of jobs or devices
- `delcompjobs.sh` command: Deletes records from the database of all jobs that were submitted and have completed successfully

4.2.5.1 Using the DMSUtil.sh command

Administrators can use the `DMSUtil.sh` command to manipulate large numbers of jobs or devices. For the purpose of simplicity, performance, avoiding the need to display large numbers of objects, or other reasons, administrators do not want to handle these large amounts of data through the Device Manager console. They would rather handle them through the `DMSUtil.sh` command.

You can use the `DMSUtil.sh` command to:

- Display jobs: (`DSUtil.sh -listjobs [job filter]`)
- Cancel jobs: (`DMSUtil.sh -canceljobs [job filter]`)
- Delete jobs: (`DMSUtil.sh deletejobs [job filter]`)
- Clean up expired jobs: (`DSUtil.sh -cleanupexpiredjobs`)
- Display devices: (`DMSUtil.sh -listdevices [device filter]`)
- Delete devices: (`DMSUtil.sh -deletedevices [device filter]`)

This command does not have any particular log file, so we recommend that you redirect stdout to a file that you specify, for example:

```
DMSUtil.sh -listjobs [your_filter] >> listjobs.log 2>&1
```

You can use the `script` command to save everything displayed on your terminal. As shown in Figure 63 on page 122, typed commands and output of the commands are saved in the /tmp/joblist.out file.

```
aix2000[/usr/lpp/TivDMS/bin]#script /tmp/joblist.out
Script started, file is /tmp/joblist.out
aix2000[/usr/lpp/TivDMS/bin]#./DMSUtil.sh -listjobs DEVICE_CLASS_NAME=Palm

******* Some data are deleted *****************
There were 37 submitted jobs matching the filter criteria.
JOB_ID = 1000085
    JOB_TYPE = DEVICE_CFG
    REALM = null
    OFFERING = null
    TARGET_DEVICE_ID = 1000061
    TARGET_DEVCLASS_ID = 1000000
    SUBMITTED_TIME = Tue Nov 14 15:45:09 JST 2000
    ACTIVATION_TIME = Fri Jan 01 00:00:00 JST 1999
    EXPIRATION_TIME = Tue Dec 31 00:00:00 JST 2002
    JOB_STATUS = EXECUTABLE
    JOB_PRIORITY = 5
    JOB_DESCRIPTION = Device Config job for Palm named 1115_5
    JOB_INTERVAL = 0
    JOB_INTERVAL_UNIT = null
    LAST_MODIFIED = Mon Oct 30 18:10:08 JST 2000

******* Some data are deleted *******************
aix2000[/usr/lpp/TivDMS/bin]#exit
Script done, file is /tmp/joblist.out
aix2000[/usr/lpp/TivDMS/bin]#
```

Figure 63. Example of the /tmp/joblist.out file

The `script` command makes a typescript of everything displayed on your terminal. The typescript is written to the file specified by the File parameter. If no file name is given, the typescript is saved in the current directory with the file name typescript. This command ends when the forked shell exits.

You can use filter attributes to specify the results you want to display. The attributes you can use depend on which option you use with the `DMSUtil.sh` command.

Job filter

Job filter allows the following attributes:

- **JOB_TYPE**: Specifies the type of job that you want to filter by. Valid values for job type include DEVICE_CFG for device configuration jobs, SW_DIST for software distribution jobs, and RESTPAGE_MGMT for rest page management jobs. Your site may provide additional job types that you can filter by.
- **REALM**: Specifies a Tivoli Personalized Services Manager-defined realm that you want to filter by. *Realms* distinguish one service provider from another. For example, there may be a realm defined as COFFEESHOP and another realm defined as BAKERY.

- **DEAL**: Specifies a Tivoli Personalized Services Manager-defined deal that you want to filter by. A deal defines the subscriber's usage fees, special access features, and so on. Typically, a service provider offers several deals to its subscribers.
- **DEVICE_NAME**: Specifies that you want to filter by the name of the device on which the job was submitted to run.
- **DEVICE_CLASS_NAME**: Specifies that you want to filter by the name of the device class on which the job was submitted to run.
- **JOB_STATUS**: Specifies that you want to filter by the current processing status of an existing job. Status is calculated by comparing the current date and time on the server with the Device Manager database to the activation and expiration dates and times stored in the Device Manager database itself.
- **JOB_PRIORITY**: Specifies that you want to filter by the numeric level of importance the administrator attached to expediting the submitted job. A valid job priority is a number from one to ten, where one is the highest priority and ten is the lowest priority.

Device filter

The device filter allows the following attributes:

- **REALM**: Specifies a Tivoli Personalized Services Manager-defined realm that you want to filter by. Realms distinguish one service provider from another.
- **MODEL**: Specifies that you want to filter by the model number for the device. The model number can be up to 32 characters long. Blanks and special characters are allowed, except for asterisks (*).
- **DEVICE_CLASS_NAME**: Specifies that you want to filter by the name of the device class for the device.

4.2.5.2 Using the delcompjobs.sh command

Every time a new device is enrolled, the enrollment server in the Subscription Manager component submits a device configuration job to configure the new device. Over time, as these jobs are processed, thousands of records of successfully completed jobs can accumulate in the Device Manager database.

The `delcompjobs.sh` command is useful to delete records from the database of all jobs that were submitted and have completed successfully. As a result, the records of any jobs with a job status of Completed are removed.

You can also use the `DMSUtil.sh -deletejobs` command to remove completed jobs with the same effect as if you used the `delcompjobs.sh` command.

The `delcompjobs.sh` command must be run from a Device Manager server.

After you enter the command, you are asked whether you want to continue and delete the job records, or exit. Specify an `r` (or `R`) after the command if you do not want this prompt to be displayed.

Figure 64 shows an example of the `delcompjobs.sh` command on the AIX system.

In this example, we ran the `delcompjobs.sh` command first without specifying the `r` option, and we received a message. Then, we pressed `c`. The `delcompjobs.sh` command ended without doing anything. Next, we ran the `delcompjobs.sh command` with the `r` option specified. The `delcompjobs.sh` command deleted jobs without being prompted.

```
aix2000[/usr/lpp/TivDMS/bin]#./delcompjobs.sh

DYM4055I This program deletes all jobs that were submitted for single devices and have completed with a
job progress of 'OK'.
Enter Y to continue and anything else to exit.
c
aix2000[/usr/lpp/TivDMS/bin]#./delcompjobs.sh r

Deleted Jobs
------------
1000000,'DEVICE_CFG','','',1000000,1000000,'Oct 30, 2000','Jan 1, 1999','Dec 31, 2002','M',5,'Device
Config job for Palm named TEST',0,'','Oct 30, 2000'
1000001,'DEVICE_CFG','','',1000001,1000000,'Oct 30, 2000','Jan 1, 1999','Dec 31, 2002','M',5,'Device
Config job for Palm named 10GB18507807',0,'','Oct 30, 2000'
                              ----------------

                              ----------------
1000064,'DEVICE_CFG','','',1000055,1000000,'Nov 1, 2000','Jan 1, 1999','Dec 31, 2002','M',5,'Device Confi
job for Palm named EM_ID_31111',0,'','Nov 1, 2000'
1000077,'DEVICE_CFG','','',1000002,1000000,'Nov 2, 2000','Nov 2, 2000','Dec 31, 2000','M',5,'Dev Config
for EM_IS_SAMPLE',0,'','Nov 2, 2000'

Total Number of records deleted = 100.
aix2000[/usr/lpp/TivDMS/bin]#./delcompjobs.sh
```

Figure 64. Example of the delcompjobs.sh command on AIX

4.2.6 Device Manager server log files

The administrator should monitor the Device Manager server log file /var/dms/DMSMsg<n>.log.

In this file, *<n>* indicates the number of a message log that wraps between files numbered 1, 2, and 3. This log contains important error and other messages.

Other error and informational messages that the administrator should monitor are the default IBM WebSphere Application Server log files:

- /usr/WebSphere/AppServer/logs/DMS_stdout.log
- /usr/WebSphere/AppServer/logs/DMS_stderr.log

You can change file names by changing standard output and standard errors of DMS_AppServer using the IBM WebSphere Application Server Administrative Console. After you change file names, you need to restart the Device Manager server servlet to apply the changes you made. The size of these two files increase, so you need to watch them.

In addition to the previously mentioned log files, the administrator should pay attention to the following IBM HTTP Server log files:

- /usr/HTTPServer/logs/access_log
- /usr/HTTPServer/logs/error_log

4.3 Device Manager console tasks

In this section, we describe the tasks that can be performed using the Device Manager console, which runs on a Microsoft Windows client. The console is very useful for an administrator to manage pervasive devices and related jobs.

The following three functions are frequently used Device Manager console functions:

- Managing devices
- Managing jobs
- Managing Software

Detailed information about these three functions are provided in 4.3.3, “Managing devices” on page 131, in 4.3.4, “Managing jobs” on page 135, and in 4.3.5, “Managing software” on page 139.

4.3.1 Device Manager console installation

To use Device Manager console (DM console), you must first install and configure it on a Microsoft Windows client. In this section, we describe how to install DM console and start it.

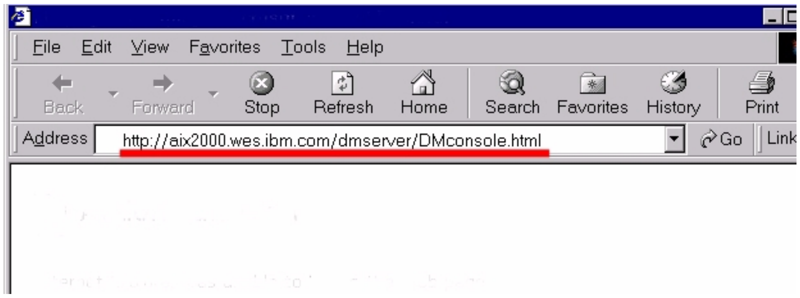
After DM server has been successfully configured and started, Device Manager console can be installed to a Microsoft Windows client. To install it, a client machine needs a \temp (or \tmp) directory with 11 MB of free disk space for the install image, plus another 32 MB for the Device Manager Console Installer and the Device Manager console. The 11 MB install image is deleted following the console installation.

For detailed information about prerequisites for the DM console machine, refer to the online manual *Tivoli Internet and Personalized Services Manager: Planning and Installation*.

Device Manager server is assumed to be configured and started in advance of Device Manager console installation.

1. From the Microsoft Windows client, start the Web browser. This case uses Microsoft Internet Explorer 5.5.
2. Open the following URL as shown in Figure 65.

   ```
   http://dmserver_hostname/dmserver/DMconsole.html
   ```

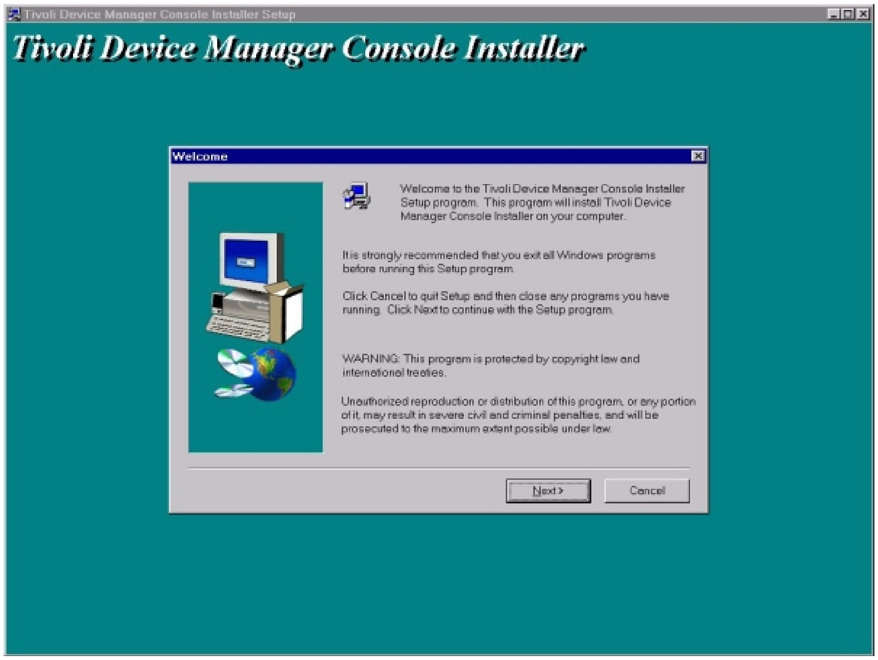


http://aix2000.wes.ibm.com/dmserver/DMconsole.html

Figure 65. Opening the DM console URL

   At the first connection to the DM console URL, Device Manager console is automatically downloaded and installed to the client machine.

3. Select the default for all the questions from the installation program, shown in Figure 66. Then the installation program completes.



Tivoli Device Manager Console Installer Setup
Tivoli Device Manager Console Installer
Welcome

Welcome to the Tivoli Device Manager Console Installer Setup program. This program will install Tivoli Device Manager Console Installer on your computer.

It is strongly recommended that you exit all Windows programs before running this Setup program.

Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.

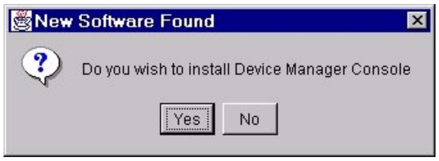
WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.

Figure 66. Device Manager console installer

Since more files need to be downloaded from the Device Manager server, installation continues.

4. Click **Yes** to install Device Manager console when the confirmation display shown in Figure 67 appears. Then, the installer starts downloading.



Do you wish to install Device Manager Console

Figure 67. Confirmation window

After downloading all the files, the Device Manager console window, shown in Figure 68 on page 128 appears. You are asked for your administrator user ID and password. For information about how to set up administrator authentication, refer to 3.4.5.3, “Authentication.properties” on page 102.

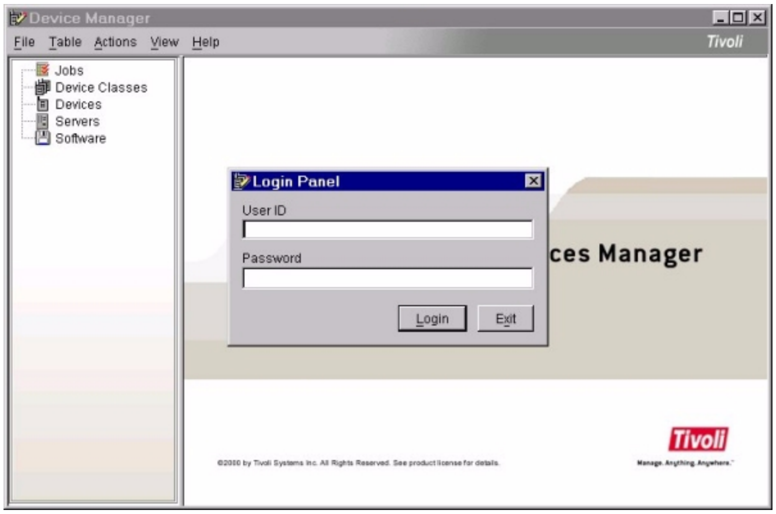

Device Manager
File Table Actions View Help
Tivoli
Jobs
Device Classes
Devices
Servers
Software
Login Panel
User ID
Password
Login
Exit
ces Manager
©2000 by Tivoli Systems Inc. All Rights Reserved. See product license for details.
Tivoli
Manage. Anything. Anywhere.

Figure 68. Device Manager console

The Device Manager console icon shown in Figure 69 is created on the desktop. Next time, you can invoke Device Manager console from this icon, or you can also invoke Device Manager console by accessing the following URL from the Web browser:

```
http://dmserver_hostname/dmserver/DMconsole.html
```

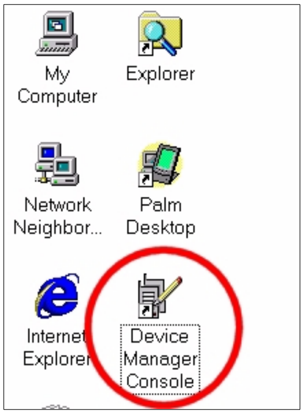


Figure 69. Device Manager console icon

Device Manager console is installed in the DM console folder in the system drive by default.

When the console starts for the first time after installation, it tries to retrieve the Transaction.properties from the device management server servlet. The Transaction.properties provides the console with information required to connect to the Device Manager database. If the Web server or application server is down, or the servlet is not running for any reason, the Transaction.properties file cannot be retrieved, and the console is unable to communicate or exchange data with the Device Manager database.

If the file is retrieved, it is saved in the local classpath on the Microsoft Windows client. The next time the console starts, it again tries to retrieve the Transaction.properties file from the DMS servlet. If the file cannot be retrieved, the console checks to see if the file was saved in the classpath of the client computer. If the file is located, the console uses the file. In this case, even if the application server or the DMS servlet is down when the console starts, it can still communicate with the database and exchange data.

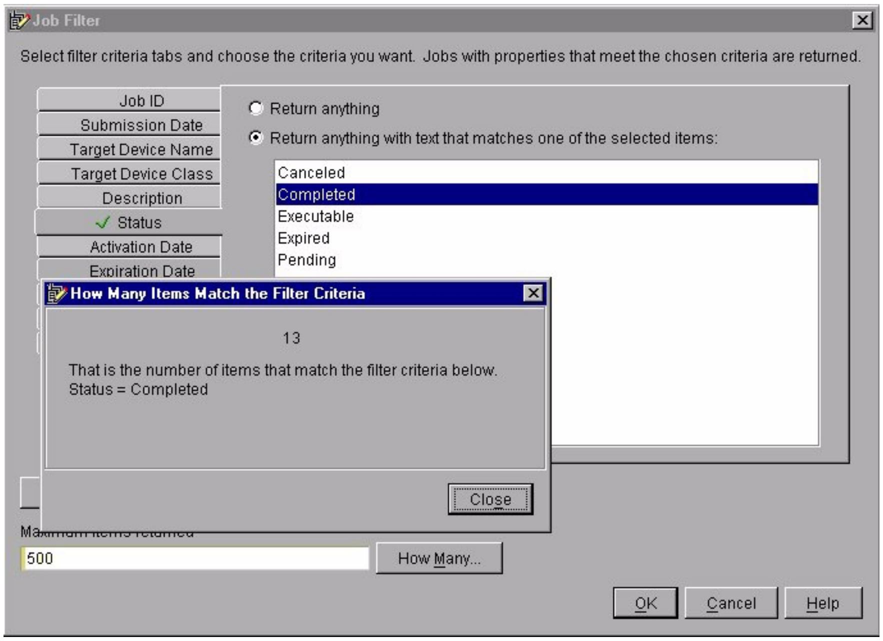
4.3.2 Filtering

If you do some management tasks without using filters, a large list of objects from the Device Manager database may be returned resulting in processing and network overhead. You can bypass or disable filtering. However, if your site supports thousands of devices, we recommend that you filter the objects you want to display.

Figure 70 on page 130 shows a filtering example. A filter is displayed by default when you select Devices, Jobs, or Software in the left pane of the console. On the filter pane, you can specify the number in the Maximum items returned field for the most database query items you want the Device Manager to display. If you don’t specify the number, the default value is 500.

The How Many button returns the number of existing items that match your current filter criteria.

Figure 70 on page 130 shows the result of the How Many button using Job Filter with JOB_STATUS = Completed. This result means that you have 13 completed jobs.



Job Filter
Select filter criteria tabs and choose the criteria you want. Jobs with properties that meet the chosen criteria are returned.
Job ID
Submission Date
Target Device Name
Target Device Class
Description
✓ Status
Activation Date
Expiration Date
○ Return anything
● Return anything with text that matches one of the selected items:
Canceled
Completed
Executable
Expired
Pending
How Many Items Match the Filter Criteria
Close
Maximum items returned
500
How Many...
OK
Cancel
Help

13

That is the number of items that match the filter criteria below.
Status = Completed

Figure 70. Job Filter

You can use filters with devices, jobs, and software. In the following section, we describe the various types of filters.

4.3.2.1 Job filter

The job filter includes the following attributes:

- Job ID
- Submission Date
- Target Device Name
- Target Device Class
- Description
- Status
- Activation Date
- Expiration Date
- Target Realm
- Target Deal
- Job Type

4.3.2.2 Device filter

The device filter includes the following attributes:

- Device Name
- Device Class
- Friendly Name
- Owner
- Realm

4.3.2.3 Software filter

The software filter includes the following attributes:

- Software Name
- Version
- Device Class

4.3.3 Managing devices

The Device Manager console uses the device management API and Device Manager database to retrieve, store, delete, view, and filter the information that the console displays and manages. This Device Manager database contains table entries that include:

- Devices
- Classes of devices
- Related parameters
- Installed software
- Device jobs
- Device Manager servers

For more information about table contents, refer to 5.1, “Data model” on page 146.

Using the device management API, the Device Manager console can access the Device Manager database directly without connecting to a Device Manager server at all, except when it displays device parameters or receives updates at session startup.

4.3.3.1 Enrolling new devices

Devices can be enrolled automatically by an enrollment server, or manually by a Device Manager administrator. The manual enrollment is not recommended, especially in any up-and-running service provider environment.

Before enrolling a new device, administrators need to install the device plug-in for that device and to configure and define its device class in the Device Manager database.

It is not recommended to enroll a new device when its device class is not configured in the Device Manager database. This could cause unpredictable results, for example, loss of data integrity.

For more information on installing the device plug-ins and configuring the device class, refer to 3.4, “Configuration” on page 80.

4.3.3.2 Adding new devices

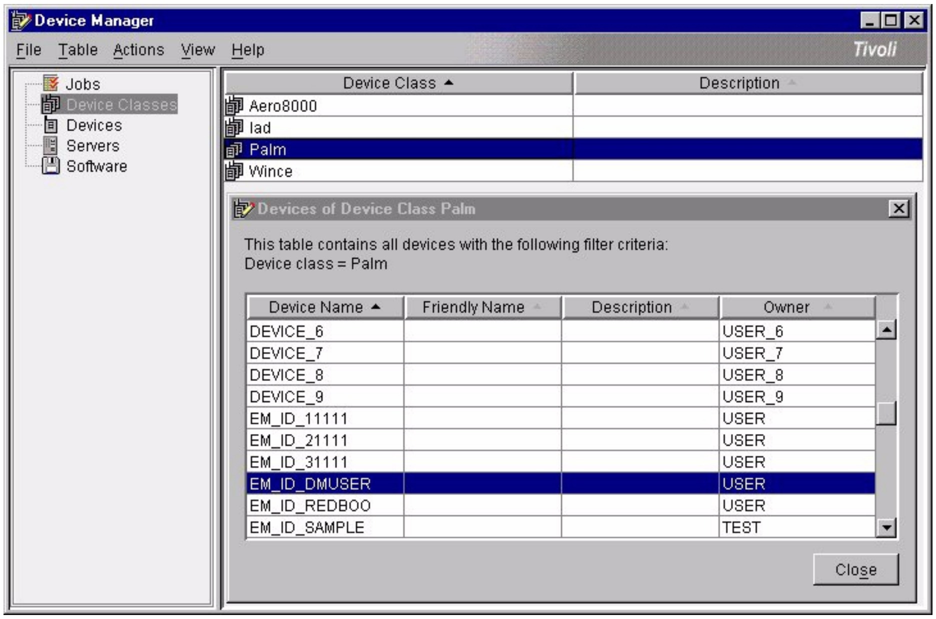
You can add a new device through the Device Manager console, but this operation is not recommended because a device name must be unique in its device class. When you name the device, you must use the exact naming convention provided by the device class. To prevent the conflict, you should enroll your device through an enrollment server.

4.3.3.3 Displaying devices

You can display the status of devices already enrolled. We recommend that you use the device filter to reduce the volume of result displayed. To display devices, select **Devices** in the left pane. The Device Filter is displayed. Specify your filter criteria and click **OK**.

You can also display devices that have installed specific software or belonged to the device class in a secondary window. For example, from the Device class window, select the target device class and right-click it, then select **View Devices.** The Device Filter is displayed. Specify your filter criteria and click **OK**.

Figure 71 shows the device information belonging to the Palm device class in a secondary window.



Device Class	Description
Aero8000	
Iad	
Palm	
Wince	

This table contains all devices with the following filter criteria:
Device class = Palm

Device Name	Friendly Name	Description	Owner
DEVICE_6			USER_6
DEVICE_7			USER_7
DEVICE_8			USER_8
DEVICE_9			USER_9
EM_ID_11111			USER
EM_ID_21111			USER
EM_ID_31111			USER
EM_ID_DMUSER			USER
EM_ID_REDBOO			USER
EM_ID_SAMPLE			TEST

Figure 71. Display devices in a secondary window

A secondary window cannot be used to modify or perform an action on a device. However, you can view devices without replacing the current display in the console.

4.3.3.4 Modifying device characteristics

You can modify the device characteristics for one or more devices, or all devices of a device class. A device and a device class have two types of characteristics: device parameters and device properties.

The difference between device parameters and device properties are:

- Device parameters depend on the class of device to which the device belongs, and they are determined by the configuration template file for that class of device.
- Device properties are common set of attributes for all devices regardless of the device class to which they belong. Device properties include information such as device class name or serial number.

The difference between device class parameters and device class properties are:

- Device class parameters are determined by the configuration template file for that class of device.

- Device class properties for all classes of a device have the same attributes, such as device class name or enrollment URL, although the values for these attributes are different for each device class.

Device parameters are initially inherited from the device class, but can be overridden for an individual device. If a device parameter is overridden, the associated check box for the parameter value is marked in the Device Parameters window for the device. If a device parameter is inherited, the associated check box for the parameter value is not marked.

You can also modify the device name field, but we do not recommend this operation. You can use this capability to correct a mistake that has been made during a manual enrollment process, which is also not recommended. Remember that Device Manager no longer recognizes the device as enrolled if you change the device name.

The device name and device class value pair identify a unique device. Therefore, no two devices have the same combination of values in both their device name and device class values. Unique devices include:

- 10GB18507808, Palm
- 10GB18507809, Palm
- 10GB18507808, Aero8000
- 10GB18507809, Aero8000

To modify device configurations, first set the device parameters for each device class or individual device as desired and then submit a device configuration job. The actual steps are outlined in 7.4, “Device configuration” on page 220.

4.3.3.5 Deleting devices

You can delete one or more devices using the Device Manager console. When you use the console to delete a device, the following information for the deleted device is removed from the Device Manager database:

- All device parameter values set for the deleted device
- All job progress records pertaining to the deleted device
- All records of jobs submitted only to the deleted device
- All records of software that Device Manager distributed to the deleted device

To delete a device or devices, select one or more target devices. Then, right-click the selected devices, select **Delete**, and click **Yes**.

You cannot delete a device class using the Device Manager console. If you want to delete it, use the devclasscfg.sh command from the Device Manager server. For more information about the devclasscfg.sh command, refer to 4.2.2.2, “Deleting a device class” on page 114.

4.3.4 Managing jobs

The Device Manager console provides windows for manipulating job-related information in the Device Manager database. Through the console, you can submit, display, cancel, and delete jobs, as well as query the current status of jobs and view their properties.

4.3.4.1 Job types

A job is any specialized processing initiated through Device Manager or its API and performed on a device or more than one device in the same device class. When the administrator creates a job, the type of job is identified. The job types supplied with Device Manager are:

- **Device configuration**: Updates the configuration of devices, including device parameters
- **Software distribution**: Distributes new or updated software applications to devices
- **Rest page management**: Updates the rest pages for Internet appliances with, for example, timely new information from the service provider

Jobs are submitted for processing by an administrator using the Device Manager console or by another application using the device management API.

At the time the job is submitted, the administrator or application specifies the job type, any job-specific parameters, the devices the job should run on, the activation and expiration time for the job, and other specific job properties. The job type and the devices should be configured in the Device Manager database before you submit a job.

If you need to add new job types, they can be added using the jobclasscfg.sh command as shown in Figure 72 on page 136. This operation must be performed from the Device Manager server. For more information on adding a new job class, refer to 4.2.3.1, “Installing a job class” on page 116.

```
# /usr/lpp/TivDMS/bin/jobclasscfg.sh -add -type RESTPAGE_MGMT -javaclass  \
com.tivoli.dms.plugin.pvc.iad.IadSampleRestPageJob -deviceclass Iad
#
#sqlplus stage_user/oracle
SQL> select * from JOB_CLASS_TABLE where JOB_TYPE='RESTPAGE_MGMT';

JOB_TYPE                                                         DEVICE_CLASS_ID
---------------------------------------------------------------- ---------------
REALM
--------------------------------
JOB_JAVA_CLASS
--------------------------------------------------------------------------------
LAST_MODI
---------
RESTPAGE_MGMT                                                            1000003

com.tivoli.dms.plugin.pvc.iad.IadSampleRestPageJob
30-OCT-00
```

Figure 72. Example of the jobclasscfg.sh command

4.3.4.2 Job status

You need to know the status of your jobs to manage them. A submitted job may have a status of:

- **Pending**: The job has been submitted but has not reached activation data and time. You can cancel a pending job.
- **Executable**: The job has been submitted and has reached the activation data and time, but has not reached the expiration date and time. You can view the job progress of an executable job to obtain more information about it. You can cancel an executable job.
- **Completed**: The job has been completed. This job status only applies to jobs that were submitted for a single device. You can either cancel or delete a completed job.
- **Expired**: The job has passed its expiration date and time. You can view the job progress of an expired job to obtain more information about it. You can cancel or delete an expired job.
- **Canceled**: The job has been canceled from the console. You can view the job progress of a canceled job to obtain more information about it. You can delete a canceled job.

Figure 73 shows the job status flow.

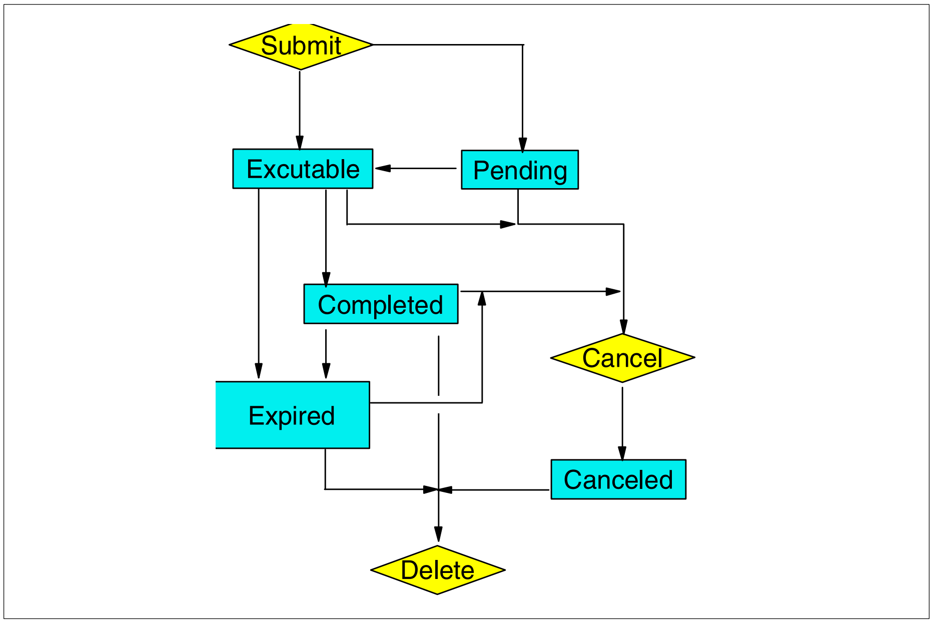


Figure 73. Job status flow

4.3.4.3 Submitting jobs

You can submit a job for:

- All devices
- All devices in one or more device classes
- One or more selected devices

When you submit a job to more than one device or device class, a job with the same properties (job type, priority, activation and expiration date, and so on) are created for every device or device class specified in the Submit Job window. These jobs are identical except for their target devices or device classes. Each job receives its own job ID and can be viewed, canceled, deleted, or otherwise managed as a separate entity using the Device Manager console.

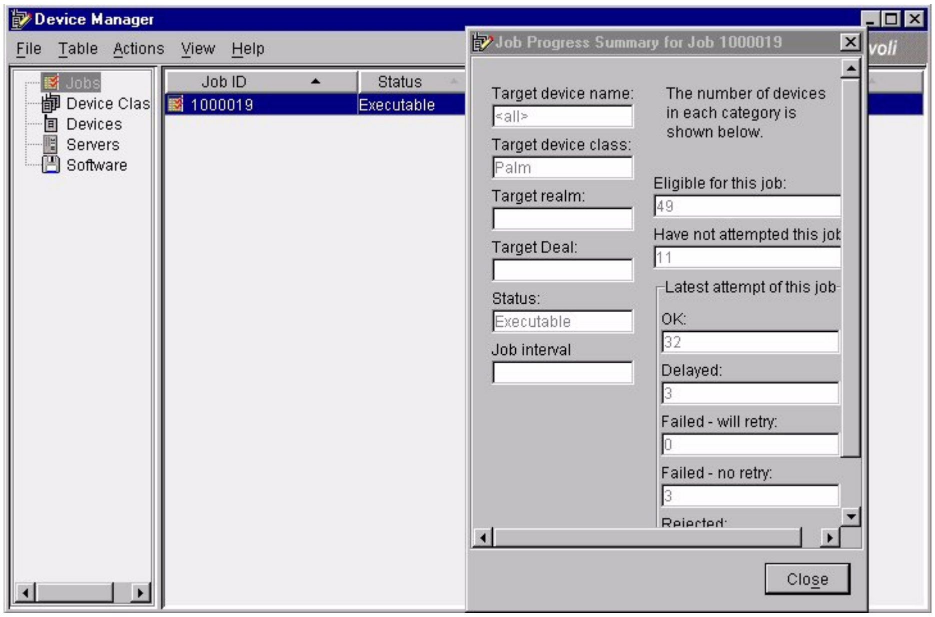
4.3.4.4 Displaying jobs

After you submit more than one job, you can display each job status, or job progress summary by using Device Manager console. It includes:

- Job status
- Job properties

- Job progress
- Job progress summary

Figure 74 shows the Job Progress Summary view.



Job ID	Status
1000019	Executable

Target device name: <all>

Target device class: Palm

Target realm:

Target Deal:

Status: Executable

Job interval

The number of devices in each category is shown below.

Eligible for this job: 49

Have not attempted this job: 11

Latest attempt of this job

OK: 32

Delayed: 3

Failed - will retry: 0

Failed - no retry: 3

Rejected:

Figure 74. Job Progress Summary

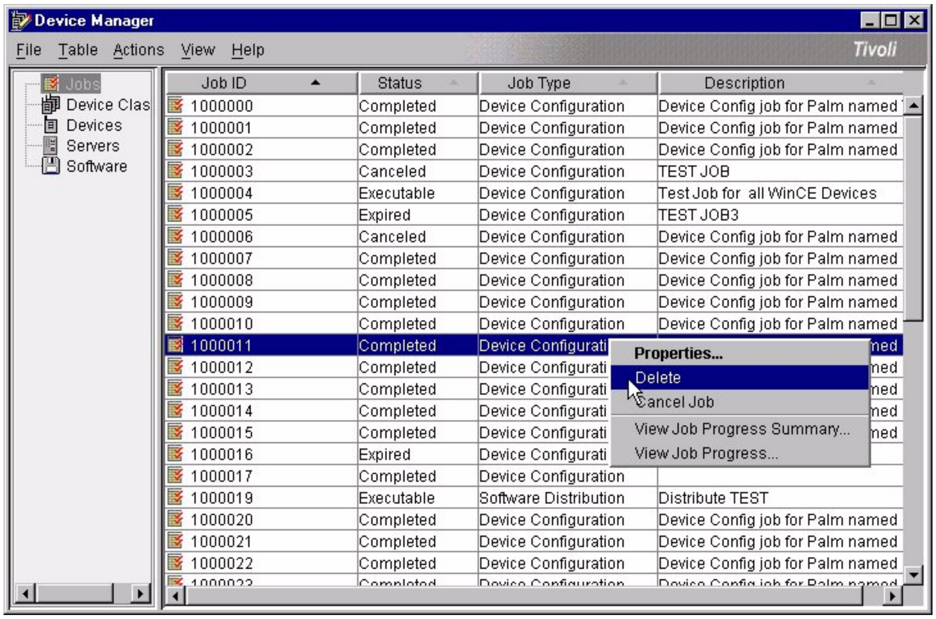
4.3.4.5 Canceling jobs

You can cancel a job when its status is Pending, Executable, Expired, or Completed. The job status changes to Canceled. After a job is canceled, it remains in the Device Manager database until you delete it.

4.3.4.6 Deleting jobs

You can delete a job only when its status is Canceled, Expired, or Completed.

Figure 75 shows the various types of job status.



Job ID	Status	Job Type	Description
1000000	Completed	Device Configuration	Device Config job for Palm named
1000001	Completed	Device Configuration	Device Config job for Palm named
1000002	Completed	Device Configuration	Device Config job for Palm named
1000003	Canceled	Device Configuration	TEST JOB
1000004	Executable	Device Configuration	Test Job for all WinCE Devices
1000005	Expired	Device Configuration	TEST JOB3
1000006	Canceled	Device Configuration	Device Config job for Palm named
1000007	Completed	Device Configuration	Device Config job for Palm named
1000008	Completed	Device Configuration	Device Config job for Palm named
1000009	Completed	Device Configuration	Device Config job for Palm named
1000010	Completed	Device Configuration	Device Config job for Palm named
1000011	Completed	Device Configurati	ned
1000012	Completed	Device Configurati	ned
1000013	Completed	Device Configurati	ned
1000014	Completed	Device Configurati	ned
1000015	Completed	Device Configurati	ned
1000016	Expired	Device Configurati	
1000017	Completed	Device Configuration	
1000019	Executable	Software Distribution	Distribute TEST
1000020	Completed	Device Configuration	Device Config job for Palm named
1000021	Completed	Device Configuration	Device Config job for Palm named
1000022	Completed	Device Configuration	Device Config job for Palm named
1000023	Completed	Device Configuration	Device Config job for Palm named

Figure 75. Job status through Device Manager console

4.3.5 Managing software

Device Manager does not maintain a library of software that you distribute. Instead, Device Manager stores URLs in the Device Manager database that point to the software that you have prepared to deploy and provides software distribution jobs you can submit to deploy the software.

4.3.5.1 Adding software

Device Manager stores URLs in the Device Manager database that point to the software you want to deploy. Before you register the software by adding software information to the database, you need to copy the software into a location accessible using HTTP or FILE protocol on a Web server where the Device Manager server can access it, and prepare a software distribution package.

Then, you can use the Device Manager console to register the software with the database. This entails creating a software definition that contains the URL that points to the software. The final step is to distribute the software to devices by submitting a software distribution job using the software definition with its URL.

4.3.5.2 Distributing software

After you complete the three steps in 4.3.5.1, “Adding software” on page 139, you can submit a software distribution job that deploys the software to one or more target devices.

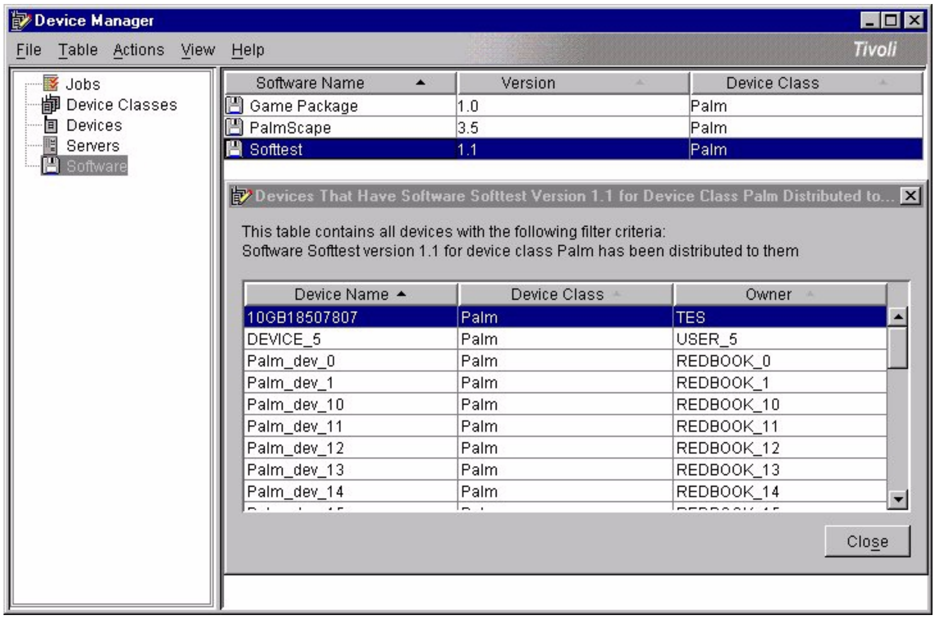
To distribute software to one or more devices whose device class supports software distribution, submit a job that specifies a job type of Software Distribution.

Refer to Chapter 7, “Using the DM functions” on page 193, for actual operations.

4.3.5.3 Displaying devices that received selected software

If you want to display the number of devices that are using selected software, or which devices already have it, you can view the list shown in Figure 76.

To display a list of software, right-click a software object in the right pane, select **View Devices** from the context menu, use **Device Filter** if you need, and click **OK**.



Software Name	Version	Device Class
Game Package	1.0	Palm
PalmScape	3.5	Palm
Softtest	1.1	Palm

Devices That Have Software Softtest Version 1.1 for Device Class Palm Distributed to...

This table contains all devices with the following filter criteria:
Software Softtest version 1.1 for device class Palm has been distributed to them

Device Name	Device Class	Owner
10GB18507807	Palm	TES
DEVICE_5	Palm	USER_5
Palm_dev_0	Palm	REDBOOK_0
Palm_dev_1	Palm	REDBOOK_1
Palm_dev_10	Palm	REDBOOK_10
Palm_dev_11	Palm	REDBOOK_11
Palm_dev_12	Palm	REDBOOK_12
Palm_dev_13	Palm	REDBOOK_13
Palm_dev_14	Palm	REDBOOK_14

Figure 76. Devices that received selected software

4.3.5.4 Deleting software

You can remove the record of a software object from the Device Manager database when you no longer want the software to be available for distribution to devices of a particular class.

When you remove the software object, the object and its properties are deleted from the Device Manager database and removed from the Device Manager console. This removal includes the URL that points to the location of the software meta file package definition file or file package definition file.

You can remove a software object only if the software is not currently listed in the software record for any existing device of the device class affiliated with the software. Before you clean up any of these records, you need to make sure that the software is no longer being used by any device in the class, and has been removed from the devices.

The following operation removes records from the Device Manager database, but does not remove any actual software and definition files from any device or server:

Software -> select your target software -> Delete -> Yes

4.4 Enabling SSL

For system administrators, system security is always important. Using Device Manager 1.1, your pervasive device can communicate to the DM server with SSL protocol. This section explains how to enable the SSL setting in both the DM Server and pervasive devices.

4.4.1 Enabling SSL on the DM server

If you want to communicate using SSL, you need to complete the following two steps. Before you start these operations, refer to 3.3.2, “Installation tasks” on page 70, for complete installation steps using the default port number = 80.

1. Make sure the SSL communication is enabled on the IBM HTTP Server by modifying the httpd.conf file to ensure secure connections over the network. For information on how to modify the httpd.conf file, refer to:

   `http://www.ibm.com/software/webservers/httpservers`

2. While you are using the `dms_install.sh` command, when you are asked to specify the port number used by *DeviceManagementServerServlet*, type `443` and press Enter instead of the default value (port number = 80) as shown in Figure 77 on page 142.

```
=========================================================================
Specify the port number used by Device Manager Server Servlet
(default port number = 80): (press Enter)
443
---------------------------------
Port number is 443
---------------------------------
```

Figure 77. Specifying the port number used by DeviceManagementServerServlet

3. As a part of the post-installation tasks, you need to configure the application server. During these steps, to identify the host name of the Device Manager server and enable physical devices to connect at server port 443 (or the port you specified for the DMS servlet when you installed the Device Manager server), from the IBM WebSphere Application Server Administrative Console, complete these steps:

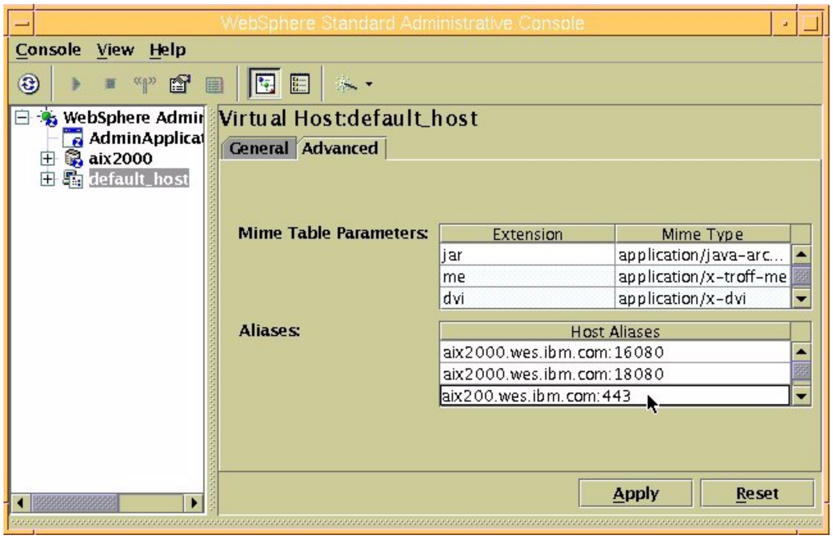
   a. From the console, click **Topology**.

   b. On the left pane, open **WebSphere Administrative Domain** in the tree.

   c. Click **default_host**.

   d. On the right pane, click the **Advanced** tab.

   e. In the Aliases list (shown in Figure 78), add a new host name entry for your Device Manager server. Make sure the port number matches the number you specified for the Device Manager server when you installed it, for example, aix2000.wes.ibm.com:443.



Virtual Host:default_host

Mime Table Parameters:

Extension	Mime Type
jar	application/java-arc...
me	application/x-troff-me
dvi	application/x-dvi

Aliases:

Host Aliases
aix2000.wes.ibm.com:16080
aix2000.wes.ibm.com:18080
aix200.wes.ibm.com:443

Figure 78. Adding an alias in the Host Aliases field

4. Click **Apply**, and then restart DMS_AppServer for the new Host Alias that you added to take effect.

4.4.2 Enabling SSL for client devices

After the Device Manager server is ready to communicate with pervasive devices using SSL protocol, you also need to enable SSL for client devices. The tasks you need to complete are:

1. Change the device properties or device class properties and submit a device configuration job to the target devices.

   You need to change the DMSPort device parameter to `443` from 80 (default) and change the SSLOn device parameter to `1` from 0 (default). Then, you need to submit the device configuration job to the target devices to affect these parameters.

   The DMSPort and SSLOn device parameters are for the Palm device. For information about each device class, refer to the online manual *Tivoli Internet and Personalized Services Manager Device Manager: Device Plug-in Notes*.

   For information on how to modify device properties, refer to 7.5.3, “Server operations” on page 226.

2. Install the certificate file to the target pervasive device.

You need to install the certificate file, which will be offered from the ISP, to pervasive devices that want to communicate with SSL protocol. This operation should be done using the HotSync operation.

For more information about how to install client software, refer to 7.1, “Device agent installation” on page 193.

Chapter 5. Device Manager database

This chapter describes the data model, tables, and columns of the Device Manager database. This information helps plug-in developers and system architects to understand Device Manager architecture. Please note that the database structure described in this chapter applies only to Device Manager 1.1. You *must not* modify any portion of the database by other methods than the supported programming interface.

Device Manager database is the core component of the Device Manager feature. This database has much information about devices, software to be distributed, and other operations required to manage devices.

The Device Manager database consists of a series of relational database tables for storing information about devices and their associated resources. As a TPSM feature, Device Manager provides its device management information to the Subscription Manager feature, which uses this device information for more personalized services.

The device management data tables are stored in the same database as that used by the Subscription Manager. Thus, TPSM features use the same database but different tables. The reason why there are no direct relationships between device management data and subscription management data is provisioning. Device management data could be used through the device management API by not only the Subscription Manager but other applications.

Important notice for this chapter

Any modifications to the Device Manager portion of the database *must* be made using the supported Device Manager programming APIs. Changes made using other mechanisms may invalidate portions of the database, breaking the installation and making it unsupportable.

The database schema described in this chapter is valid for Device Manager 1.1 only and will be changed as new software is released.

Tivoli Personalized Services Manager supports IBM DB2 Universal Database Version 7.1, and Oracle8i database 8.1.5, as its management database DBMS. Their platform can be either AIX or Sun Solaris, and they can reside on the same box as TPSM or on other boxes, depending on system requirements. There is a DB2 UDB version and an Oracle version of TPSM.

This is because there are slight differences between the scripts. However, there are no differences in the database tables, columns, and so forth.

Determine the size of the database according to the estimated number of devices. Since the Device Manager database resides in the same database as the Subscription Manager database, actual table space size is the sum of the two.

Table 8 shows the database size required by Device Manager.

Table 8. Database size required by Device Manager

Planned number of devices	Database size in bytes
100	971,715
1,000	9,341,715
10,000	93,041,715
50,000	465,041,715
100,000	930,041,715
500,000	4,650,041,715
1,000,000	9,300,041,715

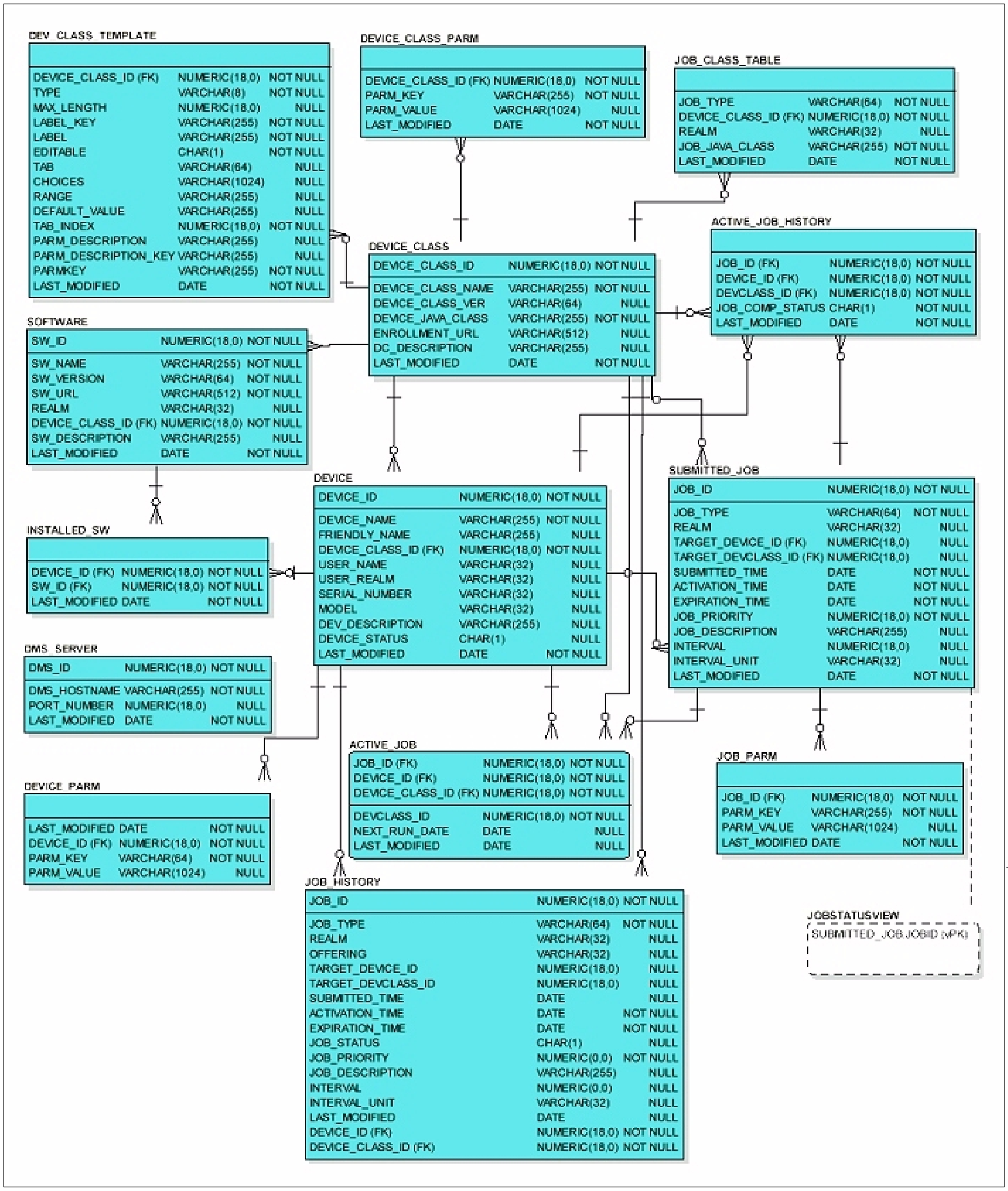
You can use the following formula to obtain the required database size, based on the planned number of devices (n):

$$\text{Database size in bytes} = (41,715 + 9,300) \times \text{n}$$

For more information about the Subscription Manager database, see the online manual *Tivoli Internet and Personalized Services Manager: Planning and Installation*.

5.1 Data model

Figure 79 shows the Device Manager database data model.



DEV_CLASS_TEMPLATE

DEVICE_CLASS_ID (FK)	NUMERIC(18,0)	NOT NULL
TYPE	VARCHAR(8)	NOT NULL
MAX_LENGTH	NUMERIC(18,0)	NULL
LABEL_KEY	VARCHAR(255)	NOT NULL
LABEL	VARCHAR(255)	NOT NULL
EDITABLE	CHAR(1)	NOT NULL
TAB	VARCHAR(64)	NULL
CHOICES	VARCHAR(1024)	NULL
RANGE	VARCHAR(255)	NULL
DEFAULT_VALUE	VARCHAR(255)	NULL
TAB_INDEX	NUMERIC(18,0)	NOT NULL
PARM_DESCRIPTION	VARCHAR(255)	NULL
PARM_DESCRIPTION_KEY	VARCHAR(255)	NULL
PARMKEY	VARCHAR(255)	NOT NULL
LAST_MODIFIED	DATE	NOT NULL

DEVICE_CLASS_PARM

DEVICE_CLASS_ID (FK)	NUMERIC(18,0)	NOT NULL
PARM_KEY	VARCHAR(255)	NOT NULL
PARM_VALUE	VARCHAR(1024)	NULL
LAST_MODIFIED	DATE	NOT NULL

JOB_CLASS_TABLE

JOB_TYPE	VARCHAR(64)	NOT NULL
DEVICE_CLASS_ID (FK)	NUMERIC(18,0)	NOT NULL
REALM	VARCHAR(32)	NULL
JOB_JAVA_CLASS	VARCHAR(255)	NOT NULL
LAST_MODIFIED	DATE	NOT NULL

ACTIVE_JOB_HISTORY

JOB_ID (FK)	NUMERIC(18,0)	NOT NULL
DEVICE_ID (FK)	NUMERIC(18,0)	NOT NULL
DEVCLASS_ID (FK)	NUMERIC(18,0)	NOT NULL
JOB_COMP_STATUS	CHAR(1)	NOT NULL
LAST_MODIFIED	DATE	NOT NULL

DEVICE_CLASS

DEVICE_CLASS_ID	NUMERIC(18,0)	NOT NULL
DEVICE_CLASS_NAME	VARCHAR(255)	NOT NULL
DEVICE_CLASS_VER	VARCHAR(64)	NULL
DEVICE_JAVA_CLASS	VARCHAR(255)	NOT NULL
ENROLLMENT_URL	VARCHAR(512)	NULL
DC_DESCRIPTION	VARCHAR(255)	NULL
LAST_MODIFIED	DATE	NOT NULL

SOFTWARE

SW_ID	NUMERIC(18,0)	NOT NULL
SW_NAME	VARCHAR(255)	NOT NULL
SW_VERSION	VARCHAR(64)	NOT NULL
SW_URL	VARCHAR(512)	NOT NULL
REALM	VARCHAR(32)	NULL
DEVICE_CLASS_ID (FK)	NUMERIC(18,0)	NOT NULL
SW_DESCRIPTION	VARCHAR(255)	NULL
LAST_MODIFIED	DATE	NOT NULL

DEVICE

DEVICE_ID	NUMERIC(18,0)	NOT NULL
DEVICE_NAME	VARCHAR(255)	NOT NULL
FRIENDLY_NAME	VARCHAR(255)	NULL
DEVICE_CLASS_ID (FK)	NUMERIC(18,0)	NOT NULL
USER_NAME	VARCHAR(32)	NULL
USER_REALM	VARCHAR(32)	NULL
SERIAL_NUMBER	VARCHAR(32)	NULL
MODEL	VARCHAR(32)	NULL
DEV_DESCRIPTION	VARCHAR(255)	NULL
DEVICE_STATUS	CHAR(1)	NULL
LAST_MODIFIED	DATE	NOT NULL

SUBMITTED_JOB

JOB_ID	NUMERIC(18,0)	NOT NULL
JOB_TYPE	VARCHAR(64)	NOT NULL
REALM	VARCHAR(32)	NULL
TARGET_DEVICE_ID (FK)	NUMERIC(18,0)	NULL
TARGET_DEVCLASS_ID (FK)	NUMERIC(18,0)	NULL
SUBMITTED_TIME	DATE	NOT NULL
ACTIVATION_TIME	DATE	NOT NULL
EXPIRATION_TIME	DATE	NOT NULL
JOB_PRIORITY	NUMERIC(18,0)	NOT NULL
JOB_DESCRIPTION	VARCHAR(255)	NULL
INTERVAL	NUMERIC(18,0)	NULL
INTERVAL_UNIT	VARCHAR(32)	NULL
LAST_MODIFIED	DATE	NOT NULL

INSTALLED_SW

DEVICE_ID (FK)	NUMERIC(18,0)	NOT NULL
SW_ID (FK)	NUMERIC(18,0)	NOT NULL
LAST_MODIFIED	DATE	NOT NULL

DMS_SERVER

DMS_ID	NUMERIC(18,0)	NOT NULL
DMS_HOSTNAME	VARCHAR(255)	NOT NULL
PORT_NUMBER	NUMERIC(18,0)	NULL
LAST_MODIFIED	DATE	NOT NULL

ACTIVE_JOB

JOB_ID (FK)	NUMERIC(18,0)	NOT NULL
DEVICE_ID (FK)	NUMERIC(18,0)	NOT NULL
DEVICE_CLASS_ID (FK)	NUMERIC(18,0)	NOT NULL
DEVCLASS_ID	NUMERIC(18,0)	NOT NULL
NEXT_RUN_DATE	DATE	NULL
LAST_MODIFIED	DATE	NULL

JOB_PARM

JOB_ID (FK)	NUMERIC(18,0)	NOT NULL
PARM_KEY	VARCHAR(255)	NOT NULL
PARM_VALUE	VARCHAR(1024)	NULL
LAST_MODIFIED	DATE	NOT NULL

DEVICE_PARM

LAST_MODIFIED	DATE	NOT NULL
DEVICE_ID (FK)	NUMERIC(18,0)	NOT NULL
PARM_KEY	VARCHAR(64)	NOT NULL
PARM_VALUE	VARCHAR(1024)	NULL

JOB_HISTORY

JOB_ID	NUMERIC(18,0)	NOT NULL
JOB_TYPE	VARCHAR(64)	NOT NULL
REALM	VARCHAR(32)	NULL
OFFERING	VARCHAR(32)	NULL
TARGET_DEVICE_ID	NUMERIC(18,0)	NULL
TARGET_DEVCLASS_ID	NUMERIC(18,0)	NULL
SUBMITTED_TIME	DATE	NULL
ACTIVATION_TIME	DATE	NOT NULL
EXPIRATION_TIME	DATE	NOT NULL
JOB_STATUS	CHAR(1)	NULL
JOB_PRIORITY	NUMERIC(0,0)	NOT NULL
JOB_DESCRIPTION	VARCHAR(255)	NULL
INTERVAL	NUMERIC(0,0)	NULL
INTERVAL_UNIT	VARCHAR(32)	NULL
LAST_MODIFIED	DATE	NULL
DEVICE_ID (FK)	NUMERIC(18,0)	NOT NULL
DEVICE_CLASS_ID (FK)	NUMERIC(18,0)	NOT NULL

JOBSTATUSVIEW

SUBMITTED_JOB.JOBID (FK)

Figure 79. Device Manager database data model

The tables in the database are:

- ACTIVE_JOB
- ACTIVE_JOB_HISTORY
- DEV_CLASS_TEMPLATE
- DEVICE
- DEVICE_CLASS
- DEVICE_CLASS_PARM
- DEVICE_PARM
- DMS_SERVER
- INSTALLED_SW
- JOB_CLASS_TABLE
- JOB_HISTORY
- JOB_PARM
- SOFTWARE
- SUBMITTED_JOB

This section describes the fields in each table. We use a dash (-) in the Field length column for the DATE Field type because this value depends on the database software in use.

5.1.1 ACTIVE_JOB table

The ACTIVE_JOB table maintains currently activated jobs. If a submitted job is targeted to multiple devices, the job is broken into multiple jobs, where each job is targeted to each device, and the job records are stored in this table. When a device connects, the Device Manager server checks this table to see if there are any jobs for the device.

Table 9 describes the fields defined in each row of the ACTIVE_JOB table. Each row corresponds to a job for a device.

Table 9. ACTIVE_JOB table

Field name	Field type	Field length	Required
JOB_ID (PK,FK)	NUMERIC	18,0	YES
DEVICE_ID (PK,FK)	NUMERIC	18,0	YES
DEVCLASS_ID (PK,FK)	NUMERIC	18,0	YES
NEXT_RUN_DATE	DATE	-	NO
LAST_MODIFIED	DATE	-	NO

Each field in Table 9 is defined here:

JOB_ID The ID of the job that is active. This ID is a foreign key that maps to the JOB_ID field of the SUBMITTED_JOB table.

DEVICE_ID The ID of the device for which this job is activated. This ID is a foreign key that maps to the DEVICE_ID field of the DEVICE table.

DEVCLASS_ID The ID of the device class to which the device-specified DEVICE_ID key belongs. This ID is a foreign key that maps to the DEVICE_CLASS_ID field of the DEVICE_CLASS table.

NEXT_RUN_DATE This field controls periodic jobs. When the periodic job is performed on a device, this field is filled with a DATE value that is calculated from the INTERVAL and INTERVAL_UNIT fields of the corresponding row in the SUBMITTED_JOB table. If this field is filled with a DATE value, this job is not performed until that particular date.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.2 ACTIVE_JOB_HISTORY table

One device could have made several attempts to run a job. All of those attempts are recorded in this table. For a periodic job, the repeat interval of the periodic job is not used in any way. Therefore, its latest attempt might be quite old, and unattempted means that the device has never attempted this job.

Table 10 describes the fields defined in each row of the ACTIVE_JOB_HISTORY table.

Table 10. ACTIVE_JOB_HISTORY table

Field name	Field type	Field length	Required
JOB_ID (FK)	NUMERIC	18,0	YES
DEVICE_ID (FK)	NUMERIC	18,0	YES
DEVCLASS_ID (FK)	NUMERIC	18,0	YES
JOB_COMP_STATUS	CHAR	1	YES

Field name	Field type	Field length	Required
LAST_MODIFIED	DATE	-	NO

Each field in Table 10 on page 149 is defined here:

JOB_ID The ID of the job which was performed. This ID is a foreign key that maps to the JOB_ID field of the SUBMITTED_JOB table.

DEVICE_ID The ID of the device for which this job is performed. This ID is a foreign key that maps to the DEVICE_ID field of the DEVICE table.

DEVCLASS_ID The ID of the device class to which the device specified DEVICE_ID key belongs. This ID is a foreign key that maps to the DEVICE_CLASS_ID field of the DEVICE_CLASS table.

JOB_COMP_STATUS This field shows the completion status of this job. Possible values are O (OK), D (Delayed), R (Failed--will retry), and N (Failed--no retry.)

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.3 DEV_CLASS_TEMPLATE table

A device class template file is provided with each device class defined to Device Manager and describes the set of device parameters required by devices of that device class. To use these device parameters, the device parameters should be registered in this table before starting the Device Manager. This task is done in conjunction with registering the device class.

Table 11 describes the fields defined in each row of the DEV_CLASS_TEMPLATE table. Each row corresponds to the definition of one device parameter in the template file for a particular device class. Device classes that define multiple parameters in their template file will have one row in this table for each parameter. The information in this table is created by the device class install utility and is used by the Device Manager console to build a graphics user interface (GUI) to allow administrators to view and modify device parameters for specific devices or device classes.

Table 11. DEV_CLASS_TEMPLATE table

Field name	Field type	Field length	Required
DEVICE_CLASS_ID (FK)	NUMERIC	18,0	Yes
TYPE	VARCHAR	8	Yes
MAX_LENGTH	NUMERIC	18,0	No
LABEL_KEY	VARCHAR	255	Yes
LABEL	VARCHAR	255	Yes
EDITABLE	CHAR	1	Yes
TAB	VARCHAR	64	No
CHOICES	VARCHAR	1024	No
RANGE	VARCHAR	255	No
DEFAULT_VALUE	VARCHAR	255	No
TAB_INDEX	NUMERIC	18,0	Yes
PARM_DESCRIPTION	VARCHAR	255	No
PARM_DESCRIPTION_KEY	VARCHAR	255	No
PARM_KEY	VARCHAR	255	Yes
LAST_MODIFIED	DATE	-	Yes

Each of the field names in Table 11 are defined here:

DEVICE_CLASS_ID The ID of the device class for which this template parameter is defined. This ID is a foreign key that maps to the DEVICE_CLASS_ID field of the DEVICE_CLASSES table.

TYPE The data type of the parameter. Support types are String, Integer, Boolean, or Tab.

LABEL_KEY The label key of the parameter. The label key is used to look up the name of the parameter in the translation resource bundle which can be displayed.

LABEL The default name of the parameter that can be displayed. This label is used as the name of the parameter if the name cannot be retrieved from the translation resource bundle using the LABEL_KEY. This field supports DBCS characters to allow maximum flexibility to device class implementations.

EDITABLE Indicates whether the parameter should be editable by an administrator (that is, whether the field should be enabled in the GUI). This field will be set to either “T” (true) or “F” (false).

TAB Defines the name of the tab under which this parameter should be grouped. For device classes that support a large number of parameters, this tab field is used to create a multi-tabbed property sheet, where each tab displays a different group of parameters. The tab name is used to look up the name of the tab that can be displayed in the translation resource bundle.

CHOICES Specifies the list of valid values, separated by commas, for the parameter. Each of these values is used to look up the name of the value that can be displayed in the translation resource bundle. This field is optional and is only used for parameters whose values must be one of a specific list of valid values.

RANGE Specifies the valid range of values. This field is optional and is only used for Integer data types whose values must be within a specific range. The format of this field is “start,end”. This field supports DBCS characters to allow maximum flexibility to device class implementations.

DEFAULT_VALUE Specifies the default value of the parameter when no value is explicitly set by an administrator. This field is optional and is only used for parameters that have a default value. This field supports DBCS characters to allow maximum flexibility to device class implementations.

TAB_INDEX Specifies the position of this parameter relative to other parameters on the same tab.

PARM_DESCRIPTION The description of the parameter or tab. This description is used if the corresponding description is not found in a bundle. It is the default description of the parameter or tab. This description is used if the description cannot be retrieved from the translation resource bundle using the PARM_DESCRIPTION_KEY. This field supports DBCS characters to allow maximum flexibility to device class implementations.

PARM_DESCRIPTION_KEY The description key of the parameter. The description key is used to look up the description in a translating resource bundle. This field is optional and supports DBCS characters to allow maximum flexibility to device class implementations.

PARM_KEY The parameter key that is sent to the device. The set of the parameter key and its value is sent to the device in the device configuration job.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.4 DEVICE table

The device table maintains a base set of information for each device being managed. Each device is associated with a single owner, such as a Tivoli Personalized Services Manager subscriber.

Table 12 describes the fields defined in each row of the DEVICE table. Each row corresponds to a single instance of a device and maintains information about that device.

The primary key is DEVICE_ID. A set of DEVICE_NAME and DEVICE_CLASS_ID is the alternate key.

Table 12. DEVICE table

Field name	Field type	Field length	Required
DEVICE_ID (PK)	NUMERIC	18,0	Yes
DEVICE_NAME	VARCHAR	255	Yes
DEVICE_CLASS_ID (FK)	NUMERIC	18,0	Yes
FRIENDLY_NAME	VARCHAR	255	No
USER_NAME	VARCHAR	32	No
USER_REALM	VARCHAR	32	No
SERIAL_NUMBER	VARCHAR	32	No

Field name	Field type	Field length	Required
MODEL	VARCHAR	32	No
DEV_DESCRIPTION	VARCHAR	255	No
DEVICE_STATUS	CHAR	1	No
LAST_MODIFIED	DATE	-	Yes

The fields in Table 12 are defined here:

DEVICE_ID This is generated for the device when the device is created. The device ID is unique across all devices in this table. This is the primary key to the device table.

DEVICE_NAME The name of the device. Each device class is responsible for the initial identification of its devices and for determining a device name that is unique across all devices of that device class. (The device class should use device-specific information, such as a serial number or MAC address, to compute this device name.) The combination of a device's device name and its associated device class name form the unique “external” device identification for the device. This indicates that the device is identified by the combination of the device name and the device type. Thus, devices with the same names can exist, if the device types are different. This field supports DBCS characters to allow maximum flexibility to device class implementations.

DEVICE_CLASS_ID The ID of the device class that supports the device. This ID is a foreign key that maps to the DEVICE_CLASS_ID field of the DEVICE_CLASS table.

FRIENDLY_NAME The friendly name of the device (optional).

USER_NAME The user name of the owner of the device.

USER_REALM The realm name to which the user belongs.

SERIAL_NUMBER The serial number of the device. This field is optional and supports DBCS characters to allow maximum flexibility to device manufacturers.

MODEL The model of the device. This field is optional and supports DBCS characters to allow maximum flexibility to device manufacturers.

DEV_DESCRIPTION A human-readable description of the device. This field is optional and supports DBCS characters to allow maximum flexibility to device manufacturers.

DEVICE_STATUS A one-character status indicator for the device. This field is intended to allow a device to exist in various states, such as “unassigned”, “in use”, or “pending deletion”.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.5 DEVICE_CLASS table

A device class represents a type of device supported by Device Manager; for example, all devices that use Palm OS. For each device class, Device Manager is configured with the device plug-in, which consists of a Java class that provides the interface to the device and a template file that describes the set of device parameters required by devices of that device class.

Table 13 describes the fields defined in each row of the DEVICE_CLASS table. Each row corresponds to a device class defined to the DMS and maintains information about that device class.

Table 13. DEVICE_CLASSES table

Field name	Field type	Field length	Required
DEVICE_CLASS_ID (PK)	NUMERIC	18,0	Yes
DEVICE_CLASS_NAME	VARCHAR	255	Yes
DEVICE_CLASS_VER	VARCHAR	64	No
DEVICE_JAVA_CLASS	VARCHAR	255	Yes
ENROLLMENT_URL	VARCHAR	512	No
DC_DESCRIPTION	VARCHAR	255	No
LAST_MODIFIED	DATE	-	Yes

The field names in Table 13 are defined here:

DEVICE_CLASS_ID An integer ID generated for the device class when the device class is defined to the DMS. The device class ID is unique across all device classes in this table. This is the primary key to the device classes table.

DEVICE_CLASS_NAME The name of the device class. Each device class is assigned a name by the device class installer when the device class is defined to the DMS. This device class name must be unique across all defined device classes. This field supports DBCS characters to allow maximum flexibility to device class installers.

DEVICE_CLASS_VER The version of the device class (optional).

DEVICE_JAVA_CLASS The fully-qualified name of the Java class that provides the interface to devices of this device class. The methods of this Java class perform device-level operations on behalf of the job classes that invoke them.

ENROLLMENT_URL The URL on the Subscription Manager enrollment server to which devices of this device class should be redirected when a new device connects that is unknown to the DMS or has no owner associated with it.

DEV_DESCRIPTION A human-readable description of the device class. This field is optional and supports DBCS characters to allow maximum flexibility to device manufacturers.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.6 DEVICE_CLASS_PARM table

This table maintains the device parameters set or all devices of a particular device class.

Table 14 describes the fields defined in each row of the device class parameters table. Each row corresponds to one parameter (key/value pair) defined for a specific device class. Device classes that have multiple parameters defined will have one row in this table for each parameter. When a device configuration job is submitted for a device, the values in the

PARM_VALUE field for the device class is set to the device, overridden by the DEVICE_PARM settings.

Table 14. DEVICE_CLASS_PARM table

Field name	Field type	Field length	Required
DEVICE_CLASS_ID (FK)	NUMERIC	18,0	Yes
PARM_KEY	VARCHAR	255	Yes
PARM_VALUE	VARCHAR	1024	No
LAST_MODIFIED	DATE	-	Yes

The field names in Table 14 are defined here:

DEVICE_CLASS_ID The ID of the device class for which this parameter is defined. This ID is a foreign key that maps to the DEVICE_CLASS_ID field of the DEVICE_CLASS table.

PARM_KEY The key (name) of the parameter. Parameter keys are stored as PARM_KEY in the DEV_CLASS_TEMPLATE table. The device parameter is uniquely identified by the combination of DEVICE_CLASS_ID and PARM_KEY.

PARM_VALUE The value of the parameter. Parameter values are always stored as strings in the device class parameters table, even though they may be defined as other data types in the template file. This field supports DBCS characters to allow maximum flexibility to device class implementations.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.7 DEVICE_PARM table

A device parameter is a key/value pair that provides configuration information to a device, such as IP Gateway or DNS Server, which the device uses. This table maintains the parameters set for specific devices.

Table 15 on page 158 lists the fields defined in each row of the DEVICE_PARM table. Each row corresponds to one parameter (key/value pair) defined for a specific device. Devices that have multiple parameters defined will have one row in this table for each parameter. If a parameter set in the DEVICE_PARM table is also set in the DEVICE_CLASS_PARM table, the value in the DEVICE_PARM is set to the device.

Table 15. DEVICE_PARM table

Field name	Field type	Field length	Required
DEVICE_ID (FK)	NUMERIC	18,0	Yes
PARM_KEY	VARCHAR	64	Yes
PARM_VALUE	VARCHAR	1024	No
LAST_MODIFIED	DATE	-	Yes

The field names in Table 15 are defined here:

DEVICE_ID The ID of the device for which this parameter is defined. This ID is a foreign key that maps to the DEVICE_ID field of the DEVICES table.

PARM_KEY The key name of the parameter. Parameter keys are stored as PARM_KEY in the DEV_CLASS_TEMPLATE table.

PARM_VALUE The value of the parameter. Parameter values are always stored as strings in the device parameters table, even though they may be defined as other data types in the template file. This field supports DBCS characters to allow maximum flexibility to device class implementations.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.8 DMS_SERVER table

This table maintains the Device Manager servers that use this database as its Device Manager database. Each record corresponds to a Device Manager server, and the record is created by the Device Manager at the first connection to this database. Each Device Manager server coordinates the execution of device management jobs scheduled for devices. Scheduled jobs are maintained in a central database.

Table 16 describes the fields defined in each row of the DMS_SERVER table. Each row corresponds to one DMS server in the network.

Table 16. DMS_SERVER table

Field name	Field type	Field length	Required
DMS_ID (PK)	NUMERIC	18,0	Yes
DMS_HOSTNAME	VARCHAR	255	Yes
PORT_NUMBER	NUMERIC	18,0	No
LAST_MODIFIED	DATE	-	Yes

Each field in Table 16 is defined here:

DMS_ID An integer ID generated when the DMS server definition is created. The home DMS server ID is unique across all DMS servers in this table. This is the primary key to the DMS servers table.

DMS_HOSTNAME The host name (or IP address) of the home DMS server.

PORT_NUMBER The port number of the Web server on the DMS server that is configured to route requests from HTTP devices to the servlet of the corresponding device class.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.9 INSTALLED_SW table

This table maintains information about what software has been downloaded to specific devices.

Table 17 describes the fields defined in each row of the installed INSTALLED_SW table. Each row corresponds to one software package installed on one device. Devices that have multiple software packages installed will have one row in this table for each installed software package.

Table 17. INSTALLED_SW table

Field name	Field type	Field length	Required
DEVICE_ID (FK)	NUMERIC	18,0	Yes
SW_ID (FK)	NUMERIC	18,0	Yes
LAST_MODIFIED	DATE	-	Yes

Each field in Table 17 on page 159 is defined here:

DEVICE_ID The ID of the device on which the software package is installed. This ID is a foreign key that maps to the DEVICE_ID field of the DEVICES table.

SW_ID The ID of the software package installed on the device. This ID is a foreign key that maps to the SW_ID field of the SOFTWARE table.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.10 JOB_CLASS_TABLE table

A job class is a Java class that provides the implementation of a high-level device management operation, such as software distribution, device configuration, or rest page management. This table maintains the set of job classes defined to Device Manager and the set of device classes that each of these job classes supports. The records should be registered to this table before the Device Manager server starts, by executing the `dms_addplugin.sh` or `jobclasscfg.sh` script in the path `/usr/lpp/TivDMS/bin` (AIX) or `/opt/TivDMS/bin` (Solaris).

Table 18 describes the fields defined in each row of the JOB_CLASS_TABLE table. Each row defines the Java class name of the job class that implements a specific job type (that is, Software Distribution, and so on) for a specific device class. For job class implementations that support more than one device class, there must be a separate entry in this table defined for each device class supported by the job class.

Table 18. JOB_CLASS_TABLE table

Field name	**Field type**	**Field length**	**Required**
JOB_TYPE	VARCHAR	64	Yes
DEVICE_CLASS_ID (FK)	NUMERIC	18,0	Yes
REALM	VARCHAR	32	No
JOB_JAVA_CLASS	VARCHAR	255	Yes
LAST_MODIFIED	DATE	-	Yes

Each field in Table 18 is explained in the following list:

JOB_TYPE The job type that is implemented by the job class. The base set of job types that will be shipped with the DMS in the initial release are Software Distribution (SW_DIST), Device Configuration (DEVICE_CFG), and Rest Page Management (RESTPAGE_MGMT.) Job classes that implement the basic function of these job types will be developed by the device class providers. However, the DMS architecture allows other job types to be defined (by device class providers, system integrators, and so on) and added to this table.

DEVICE_CLASS_ID The ID of the device class that this job class implementation supports. This ID is a foreign key that maps to the DEVICE_CLASS_ID field of the DEVICE_CLASSES table.

REALM The realm for which this job class is used.

JOB_JAVA_CLASS The fully-qualified Java class name of the job class implementation.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.11 JOB_HISTORY table

This table contains IDs for jobs that have completed on specific devices. It is used to record when a job completes successfully on a device so that the job is not executed again on the same device. Unsuccessful completions are also recorded as an aid to problem determination. Records are removed from the table when the job is deleted from the SUBMITTED_JOB table.

Table 19 describes the fields defined in each row of the JOB_HISTORY table. Each row corresponds to the job submission progress record.

Table 19. JOB_HISTORY table

Field name	Field type	Field length	Required
JOB_ID (PK)	NUMERIC	18,0	Yes
JOB_TYPE	VARCHAR	64	Yes
REALM	VARCHAR	32	No
OFFERING	VARCHAR	32	No
TARGET_DEVICE_ID	NUMERIC	18,0	No

Field name	Field type	Field length	Required
TARGET_DEVCLASS_ID	NUMERIC	18,0	No
SUBMITTED_TIME	DATE	-	No
ACTIVATION_TIME	DATE	-	Yes
EXPIRATION_TIME	DATE	-	Yes
JOB_STATUS	CHAR	1	No
JOB_PRIORITY	NUMERIC	0,0	Yes
JOB_DESCRIPTION	VARCHAR	255	No
INTERVAL	NUMERIC	0,0	No
INTERVAL_UNIT	VARCHAR	32	No
LAST_MODIFIED	DATE	-	No

Each field in Table 19 on page 161 is explained here:

JOB_ID An integer ID that represents a submitted job. This job ID represents the same job as the same ID in the SUBMITTED_JOB table. This is the primary key to the job history table.

JOB_TYPE The job type that is submitted. Job types available for this Device Manager system are registered to JOB_CLASS_TABLE at database configuration time.

REALM The realm name for which this job is submitted. This is optional.

OFFERING The offering name for which this job is submitted. This is optional.

TARGET_DEVICE_ID The device ID for which this job is submitted. This field is filled if the job is targeted to a device.

TARGET_DEVCLASS_ID The device class ID for which this job is submitted. This field is filled if the job is targeted to a device class.

SUBMITTED_TIME The time this job is submitted.

ACTIVATION_TIME The time this job will be activated. This field allows any time value if it is less than the expiration time.

EXPIRATION_TIME The time this job will be expired. This field allows any time value that is greater than the activation time.

JOB_STATUS The job progress status. This field can take values of C (CANCELED), X (EXPIRED), M (COMPLETED), and S. The value S stands for PENDING, EXECUTABLE, or EXPIRED according to the relation between current system time, the job activation time, and the job expiration time.

JOB_PRIORITY The job execution priority. This is optional. If not specified, the value is set to 5. A valid job priority is a number from 1 to 10, where 1 is the highest priority and 10 is the lowest priority.

JOB_DESCRIPTION The description for this job. This is optional.

INTERVAL The interval of this job execution. This field is used in case this job is performed periodically.

INTERVAL_UNIT The unit of the job execution interval. This field is used together with INTERVAL field. Possible values are HOURS, DAYS, WEEKS and MONTHS.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.12 JOB_PARM table

This table contains configuration parameters provided when the job is submitted. A job parameters is a set of key/value pairs for parameters to be associated with this job. The required key names depend on the job type. Records are removed from the table when the job is deleted from the SUBMITTED_JOB table.

Table 20 describes the fields defined in each row of the JOB_PARM table. Each row corresponds to the job submission progress record.

Table 20. JOB_PARM table

Field name	Field type	Field length	Required
JOB_ID (FK)	NUMERIC	18,0	Yes
PARM_KEY	VARCHAR	255	Yes
PARM_VALUE	VARCHAR	1024	No

Field name	Field type	Field length	Required
LAST_MODIFIED	DATE	-	Yes

Each of the field names in Table 20 on page 163 are explained in the following list:

JOB_ID The job ID for which the job configuration parameter is set. This ID is a foreign key that maps to the JOB_ID field of the SUBMITTED_JOB table.

PARM_KEY The key name of the configuration parameter. For example, built-in device plug-ins use DMS_URL_OF_SOFTWARE_PACKAGE key in this field, to specify the URL of the software package to be distributed.

PARM_VALUE The value that corresponds to the key in the PARM_KEY field. In the case where a row represents the URL of a software package, the actual URL is stored in this field.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.13 SOFTWARE table

This table maintains information about software that can be downloaded to specific devices.

Table 21 describes the fields defined in each row of the SOFTWARE table. Each row corresponds to the definition of one software package that can be downloaded to devices of the corresponding device class. If a software package can be run on devices of more than one device class, there must be a separate entry in this table to define the software package for each supported device class.

Table 21. SOFTWARE table

Field name	Field type	Field length	Required
SW_ID (PK)	NUMERIC	18,0	Yes
SW_NAME	VARCHAR	255	Yes
SW_VERSION	VARCHAR	64	Yes
SW_URL	VARCHAR	512	Yes

Field name	Field type	Field length	Required
REALM	VARCHAR	32	No
DEVICE_CLASS_ID	NUMERIC	18,0	Yes
SW_DESCRIPTION	VARCHAR	255	No
LAST_MODIFIED	DATE	-	Yes

The following list explains the field names in Table 21:

SW_ID An integer ID generated when the software package definition is created. The software ID is unique across all software packages in this table. This is the primary key to the software table.

SW_NAME The name of the software package. Each software package is assigned a name by the software installer when the software package is defined to the DMS. The software name, combined with the software version, fully identify the software package, and this combination must be unique for all software packages defined for a particular device class. This field supports DBCS characters to allow maximum flexibility to device class installers.

REALM The realm for which this software will be distributed.

SW_VERSION The version of the software package. Each software package is assigned a version by the software installer when the software package is defined to the DMS. The software name, combined with the software version, fully identify the software package, and this combination must be unique for all software packages defined for a particular device class. This field supports DBCS characters to allow maximum flexibility to device class installers.

SW_URL The URL of the software package file. The format and contents of this file are defined by the device class or job class that distributes the software to devices; that is, the format and contents of this file may vary from one device class to another.

DEVICE_CLASS_ID The ID of the device class for which this software package is defined. This ID is a foreign key that maps to the DEVICE_CLASS_ID field of the DEVICE_CLASSES table.

SW_DESCRIPTION A human-readable description of the software package. This field is optional and supports DBCS characters to allow maximum flexibility to device manufacturers.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.1.14 SUBMITTED_JOB table

This table contains information about jobs submitted for a specific device, all devices of a specific device class, and all devices. This table only contains PENDING, EXECUTABLE, or EXPIRED jobs. When a job is CANCELED or COMPLETED, it is removed from this table.

Table 22 describes the fields defined in each row of the SUBMITTED_JOB table. Each row corresponds to a job submitted to the specific device or to all the devices in a device class.

Table 22. SUBMITTED_JOB table

Field name	Field type	Field length	Required
JOB_ID (PK)	NUMERIC	18,0	Yes
JOB_TYPE	VARCHAR	64	Yes
REALM	VARCHAR	32	No
OFFERING	VARCHAR	32	No
TARGET_DEVICE_ID (FK)	NUMERIC	18,0	No
TARGET_DEVCLASS_ID (FK)	NUMERIC	18,0	No
SUBMITTED_TIME	DATE	-	Yes
ACTIVATION_TIME	DATE	-	Yes
EXPIRATION_TIME	DATE	-	Yes
JOB_PRIORITY	NUMERIC	18,0	Yes
JOB_DESCRIPTION	VARCHAR	255	No
INTERVAL	NUMERIC	18,0	No
INTERVAL_UNIT	VARCHAR	32	No

Field name	Field type	Field length	Required
LAST_MODIFIED	DATE	-	Yes

The field names in Table 22 are explained here:

JOB_ID An integer ID generated when the job is submitted. The job ID is unique across all jobs in this table. This is the primary key to the submitted job table.

JOB_TYPE The job type that is submitted. Job types available for this Device Manager system are registered to JOB_CLASS_TABLE at database configuration time.

REALM The realm for which this job is submitted. This is optional.

OFFERING The offering name for which this job is submitted. This is optional.

TARGET_DEVICE_ID The device ID for which this job is submitted. This field is filled if the job is targeted to a device.

TARGET_DEVCLASS_ID The device class ID for which this job is submitted. This field is filled if the job is targeted to a device class.

SUBMITTED_TIME The time this job is submitted.

ACTIVATION_TIME The time this job will be activated. This field allows any time value if it is less than the expiration time.

EXPIRATION_TIME The time this job will be expired. This field allows any time value that is greater than the activation time.

JOB_PRIORITY The job execution priority. This is optional. If not specified, the value is set to 5. A valid job priority is a number from 1 to 10, where 1 is the highest priority and 10 is the lowest priority.

JOB_DESCRIPTION The description for this job.

INTERVAL The interval of this job execution. This field is used if this job is performed periodically.

INTERVAL_UNIT The unit of the job execution interval. This field is used together with the INTERVAL field. Possible values are HOURS, DAYS, WEEKS, and MONTHS.

LAST_MODIFIED The date and time that this table row was last modified. This field is maintained automatically by the device management APIs.

5.2 Views

There are several views in the Device Manager database. This section looks at these types of views:

- JobStatusView
- EligibleView
- AllDevicesForJob
- LastJobStatus

These views are mainly used to display the job status in Device Manager console.

5.2.1 JobStatusView

This view lists the jobs that have not completed, that have completed in retry or delay state, or that have completed successfully, but that is a periodic job. The SUBMITTED_JOB table will only contain PENDING, EXECUTABLE, or EXPIRED jobs, because when a job is CANCELED or COMPLETED, it is removed from this table.

Table 23 shows the contents of the JobStatusView view.

Table 23. JobStatusView view

Field name	Original
jobID	JOB_ID from JOB_STATUS
status	Calculated by function. Possible values are: - CANCELED - EXPIRED - COMPLETED - PENDING - EXECUTABLE

5.2.2 EligibleView

This view provides a list of jobs and all devices that have run or could run this job. This is used to present a count of devices that a specific job affects.

Table 24 shows the contents of the EligibleView view.

Table 24. EligibleView view

Field	Original	Conditions
jID	JOB_ID in ACTIVE_JOB	Rows that satisfy the following two conditions: - The job for which JOB_ID exists both in ACTIVE_JOB and SUBMITTED_JOB. - The job that is active.
jType	JOB_TYPE in ACTIVE_JOB	
jRealm	REALM in ACTIVE_JOB	
offering	OFFERING in ACTIVE_JOB	
dID	DEVICE_ID in ACTIVE_JOB	
jPriority	JOB_PRIORITY in ACTIVE_JOB	
jInterval	INTERVAL in ACTIVE_JOB	
jIntervalUnit	INTERVAL_UNIT in ACTIVE_JOB	
next_run_date	NEXT_RUN_TIME in ACTIVE_JOB	

5.2.3 AllDevicesForJob

This view provides a list of jobs and the devices that have attempted to run this job, plus the last known completion status reported by the device for the job.

This view is used to present the count of devices for each completion status for a specific job_id by issuing the following SQL query:

```
select count(device_id), job_comp_status
from lastjobstatus where job_id=xxxx
group by job_comp_status;
```

Table 25 shows the contents of AllDEviceForJob view.

Table 25. AllDeviceForJob view

Field	Original	Conditions
job_id	JOB_ID in ACTIVE_JOB or ACTIVE_JOB_HISTORY	The job that has the unique combination of JOB_ID and DEVICE_ID and that is in an ACTIVE_JOB table or ACTIVE_JOB_HISTORY table.
device_id	DEVICE_ID inACTIVE_JOB or ACTIVE_JOB_HISTORY	

5.2.4 LastJobStatus

This view provides a list of the job status. If the job has been attempted multiple times, only the status of latest attempt is shown in this view.

Table 26 shows the contents of the LastJobStatus view.

Table 26. LastJobStatus view

Field	Original	Conditions
job_id	JOB_ID in ACTIVE_JOB_HISTORY	The latest job among multiple attempts of the job to a device
device_id	DEVICE_ID in ACTIVE_JOB_HISTORY	
job_comp_status	JOB_COMP_STATUS in ACTIVE_JOB_HISTORY	

Chapter 6. Application programming interface

This chapter provides an overview of the development of the device plug-in software for a new device and explains how to integrate an external application using DMS APIs.

6.1 Overview

Device Manager includes application programming interfaces (APIs) that device manufacturers and integrators can use to develop plug-in software for managing new devices and to integrate external applications for enrollment, billing, customer care, and more. Device Manager has its own set of business objects that are based on classes and methods provided by the Tivoli Personalized Services Manager Integration Toolkit.

The device management server API defines the programming interface between the device management server servlet (DMS servlet) and the device plug-ins. This interface allows device plug-ins to interact with a Device Manager server. The device management server API is a general purpose, protocol-independent interface that serves as an abstraction layer, allowing all classes of devices, though they may operate differently, to be managed in the same way by Device Manager.

Either Device Manager, or the device manufacturer or integrator, provides a device plug-in for the new device that implements the required functionality and contains logic for performing the unique management functions for the class of devices. The device management server API can be used to develop plug-ins for any new class of device--plug-ins that implement device identification, communications, and job processing for all devices in the new device class.

The device management API defines the programming interface for manipulating the device and job-related data resources stored in the Device Manager database. Resources that can be managed include database entries describing devices and classes of devices, their parameters, the device class template, software defined for and available to devices, software installed on devices by Device Manager, Device Manager servers, and job classes.

Methods are provided to create, delete, and update these entries, as well as search the database for entries matching certain criteria. These are the same types of functions that can be done by an administrator through the Device Manager console.

6.2 Components of Device Manager

To program to the APIs provided by Device Manager, you need to understand the functions of the Device Manager database, the device plug-ins, and the device management server servlet. Table 80 illustrates the DMS components.

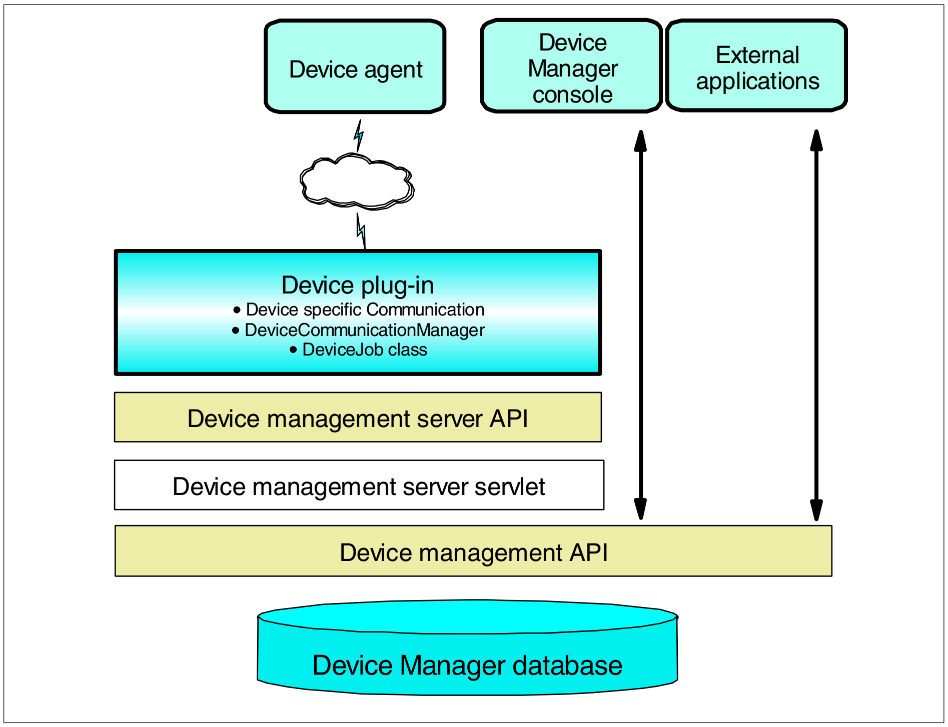


Figure 80. DMS components

6.2.1 Device Manager database

As with Tivoli Personalized Services Manager, a relational database is at the core of Device Manager. The database contains all the information needed to manage a subscriber's devices. A set of tables contains information about Device Manager jobs and devices. The APIs understand the interactions between the tables and take care of creating, updating, or deleting related information across all the tables to keep them synchronized.

The business objects of Device Manager provide a way of interacting with the database in a uniform and consistent way without having to understand SQL or the structure of the device database. These high-level, business-oriented objects are built on simpler, table-oriented functions provided by the Tivoli Personalized Services Manager integration toolkit.

6.2.2 Device plug-ins

A device plug-in resides on the Device Manager server and provides the logic that handles device identification, communications, job processing, and high-level management tasks for a particular class of managed devices. Put simply, a plug-in defines support for a Device Manager device class. A device class defines a group of devices whose operations can be similarly managed.

Plug-ins are logically structured in four pieces:

- A device-specific communication component that the plug-in software uses to communicate with the device. This interface is defined by the plug-in developer, typically it is implemented as a servlet.
- The internal device communications component, which enables communication between the plug-in software and the DMS servlet. This piece is created by the plug-in developer by providing a DeviceCommunicationManager API implementation for the plug-in.
- The set of device-specific job classes. This piece of the plug-in interacts with the DMS servlet, the device-specific communication component, and the DeviceCommunicationManager implementation. It implements the higher-level management tasks (such as software distribution or device configuration). This piece is created by the plug-in developer, by implementing the DeviceJob interface.
- An optional template file that describes the set of device parameters, such as the IP gateway or domain name server used, that are supported by all devices of the device class.

Both DeviceCommunicationManager and DeviceJob are abstract Java classes in the device management server API that the plug-in writer extends as part of developing device classes and device jobs.

6.2.3 Device management server servlet

When a device connects to the service provider's network, the device management server servlet ensures that the device is enrolled. If it is, it coordinates the processing of all scheduled device management jobs for the device. Jobs are job classes included with each device plug-in that resides on the Device Manager server. Device Manager supports job types, such as device configuration, software distribution, and rest page management.

Jobs are submitted for processing by an administrator using the Device Manager console or by another application using the device management API. At the time the job is created, the administrator or application specifies the type of job, any job-specific parameters, the devices the job should run on, the activation and expiration time for the job, and so on.

When a device connects to the service provider's network, communicating either directly or through a network dispatcher, to a Device Manager server, the DMS servlet checks to make sure the device is enrolled with the service provider and redirects it as needed:

- For a device requiring enrollment, the function redirects the device to the service provider's enrollment server for registration.
- For an enrolled device, if there are jobs pending for it to run, the function redirects the device back to the Device Manager server. This prevents the device from returning through a network dispatcher during job processing and then being rerouted to a different Device Manager server, should multiple HTTP connections be required to complete the job.

The DMS servlet searches the database for any submitted jobs eligible to be run that pertain to the device and builds a prioritized job list. To run the jobs, the DMS servlet uses device management server API calls through the device plug-in to interact with the device, often through several iterations of requests and responses.

6.2.4 Tivoli Personalized Services Manager integration toolkit

To understand the Device Manager APIs, it is necessary to have a basic understanding of the Tivoli Personalized Services Manager integration toolkit. The toolkit provides a base set of classes that are used by Tivoli Personalized Services Manager for database access. The base classes are used as the building blocks for a set of business-level classes called business objects.

These higher-level classes give the programmer a means of manipulating data and data relationships from a business perspective, rather than as bits of data in a relational database. Tivoli Personalized Services Manager integration toolkit includes a set of business objects built on the base classes for subscriber database information.

For Device Manager, the device management API uses the Tivoli Personalized Services Manager integration toolkit base classes to implement a set of business objects for device management, using the same programming model.

All of Device Manager's business objects based on the Tivoli Personalized Services Manager integration toolkit extend the integration toolkit's TxObject. Think of TxObject as an object that is manipulated in the context of a Transaction (another class provided by the toolkit). Business objects are instantiated locally for an application, and methods on the business object are invoked to set and get the values of fields in the business object. The values of these fields are then used to manipulate the relevant information in the relational database by passing the business object to methods (create, update, and so on) of a Transaction object. The Transaction object invokes the corresponding method of the business object to perform the requested operation.

For details of the integration toolkit's programming model and classes, refer to the online manual *Tivoli Internet and Personalized Services Manager Documentation: Programmer's Guide*.

6.3 Device management server API

Device plug-ins provide the device-dependent logic to perform management operations (or jobs) on specific types of devices. The communications between a device plug-in and the actual device is device-dependent and is determined by the integrator who writes the plug-in code. The device management server API defines the programming interface between a device plug-in residing on Device Manager and the rest of Device Manager (including its redirection function). The device management server API is designed to be generic and protocol-independent to support many different kinds of devices and jobs.

The device management server API includes two main abstract classes called *DeviceCommunicationManager* and *DeviceJob*. A plug-in developer extends these abstract Java classes to enable a device to be managed by Device Manager and to implement device class-specific jobs.

The following sections highlight some of the main classes in the device management server API.

6.3.1 Device and job classes

Device Manager uses the concept of device class to describe a group of devices of similar characteristics or functions. For example, a device class could include all screenphones of a particular model from a particular manufacturer. Device Manager stores information about classes of devices in its device database.

For a device class, a plug-in writer defines job classes that are supported for that device. Note that job classes are device class-specific. For example, the Device Manager product package includes software distribution for specific devices, such as Palm III and Palm V-series Computing devices. Even though both jobs can generically be described as “software distribution”, the implementation varies according to the different device class.

6.3.1.1 DeviceCommunicationManager class

The DeviceCommunicationManager abstract class has the function to enable a device to be managed by Device Manager. The interfaces defined by this class include:

- A method for instantiating a device class.
- A method for the Device Manager server to register itself as the default event handler for a device class.
- Methods that the Device Manager uses to cause the device class to redirect a device to a server. A device can be redirected to the Tivoli Personalized Services Manager enrollment application if it needs to enroll. It can also be redirected to bypass the network dispatcher and connect directly to a specific Device Manager, an important capability in a situation where the device needs to have multiple connections with the same Device Manager machine (such as a large or multi-step job).
- Methods for a device class to notify the Device Manager server of incoming events from a device. The definition of the methods describes the types of events that can occur and the parameters that must be provided for each event type. For example, for CONNECT events, the device class must provide the unique identification of the connecting device.
- A method for a device class to notify the device management server of an exception condition.

6.3.1.2 DeviceJob class

The DeviceJob abstract class has the basic function to allow device-specific jobs to be created and managed by Device Manager.

The interfaces defined by this class include:

- A method for the DMS servlet to initialize a job and enumerate any device-specific configuration or job parameters, that is, initJob().
- A method for the DMS servlet to start or resume processing of a job on a particular device. The definition of this method describes how job and

device parameters are passed to the job class and the return indicators that can be received from the job class, that is, doJob().

The DeviceJob class has several reserved parameters defined for use as job or device parameters to be passed on a call to the initJob method of DeviceJob. The reserved parameters are easily identified because their names start with <DMS_>. The parameters include networking information such as a device's default gateway, IP address, or subnet mask, and user information such as a user ID or password.

The API includes three job-specific classes that extend DeviceJob. These classes are provided as a convenience for doing some common jobs and they will handle some pre- and post-job processing tasks automatically. The classes are:

- DeviceConfigurationJob
- SoftwareDistributionJob
- RestPageManagementJob

6.3.1.3 PervasiveDeviceID class

The PervasiveDeviceID class provides an identifier to uniquely identify a specific device within Device Manager. The class contains the following information:

- The fully-qualified Java class name of the class managing the device; for example, com.tivoli.dms.deviceCommunicationManager.device.
- The short class name that uniquely identifies the type of device managed by a particular DeviceCommunicationManager, typically just the device class name.
- A unique identifier, such as the device's serial number.

6.3.1.4 DeviceManagementEvent class

The DeviceManagementEvent class provides the base function that DeviceCommunicationManagers use to alert Device Manager of device initiated events, such as a request for work. The fire() methods in the DeviceCommunicationManager class are used to initiate these events.

The following four subclasses extend this base class with event-specific functionality:

- **DeviceConnectionEvent** notifies Device Manager that a device has initiated or closed a connection.

- **DeviceRequestWorkEvent** notifies Device Manager that a device is requesting a new job or is asking to continue processing of a job in progress.
- **DeviceJobProcessingCompleteEvent** notifies Device Manager of the completion of a job and the status of the job, for example, that it completed successfully, that is failed but should be retried, and so on.
- **DeviceErrorEvent** notifies Device Manager of an error. It returns the Java exception, if there is one, as well as a description of the event.

The API includes four corresponding “listener” classes, one for each of the event-specific classes listed here, that process the events as they are received. These listeners are registered by calling methods in the DeviceCommunicationManager class.

6.3.2 Developing support for devices

To support a new device, it is the integrator's responsibility to create a device class plug-in to support the associated device class.

Before a device can be managed by Device Manager, plug-in developers must determine the best way to interact with Device Manager. You should consider the following points:

- How should the device communicate with Device Manager?
- What jobs are supported by the device?
- What steps are needed for the device to be managed by Device Manager?
- Are there device-specific parameters needed to operate the device?

Each one of these design points can take considerable developer effort.

To support a particular device class, a plug-in writer needs to complete the following steps:

1. Create a servlet (or other mechanism) that communicates with the device using HTTP or some other protocol and that communicates with Device Manager using the device management server API.
2. Optionally extend the DeviceCommunicationManager class to implement methods for managing communications.
3. As appropriate, extend the DeviceConfigurationJob, SoftwareDistributionJob, and RestPageManagementJob classes to create jobs.
4. Extend the DeviceJob class to create jobs that are specific to the device class.

5. Optionally, create a template file that defines device parameters that an administrator will use to manage the device.

To communicate with a device, providing a device agent code is required. But the design of agent code is not discussed in this book because it is specific to the device.

6.3.2.1 Developing the plug-in

A plug-in developer must determine the communication protocol between an agent program and a device plug-in. The communications between a device plug-in and the actual device is device-dependent. This component is developed by the plug-in developer, and is typically implemented as a servlet.

The above device-specific communication component is the client side code needed to enable the device for Device Manager. A plug-in developer also needs to develop an internal device communications component, which enables communication between the device-specific communication component and the DMS servlet. This component is created by the plug-in developer by providing a DeviceCommunicationManager API implementation for the plug-in.

A plug-in developer needs to define job classes that are supported for that device. Note that job classes are device class-specific. Job classes implement the logic for performing higher-level management tasks, such as software distribution, device configuration, and rest page management. A job class performs these high-level tasks by invoking the device class methods necessary to accomplish the task.

A plug-in developer extends DeviceCommunicationManager and DeviceJob abstract Java classes to enable a device to be managed by Device Manager and to implement device class-specific jobs.

A description of the API calls and other communications between a device and Device Manager components can be found in 2.3.2, “Job flow” on page 45.

Developing jobs for a device

To create a general-purpose job, extend the DeviceJob class. In the default constructor of a particular DeviceJob, required parameters can be defined through one or more requiredParameters.addElement(parameters), where requiredParameters is a Vector of required parameters. The DeviceJob base class automatically checks the required parameters against the job parameters and device parameters when Device Manager calls the job.performPreProcessing() method.

For example, suppose you extended DeviceJob to create a specific job called MyJob as shown in Figure 81.

```
import com.tivoli.dms.dmserver.DeviceJob;

public class MyJob extends DeviceJob {

    public MyJob {

        requiredParameters.addElement(SomeParameterMyJobNeeds);

    }

    public void doJob(Object deviceContext) throws DeviceManagementException {

        // Your job goes here....

    }
}
```

Figure 81. Example of extending the DeviceJob class

It is important to note that if there is a conflict between a device parameter and a job parameter (in other words, if the keywords match), the job parameter takes precedence and will be used to complete the job.

Job-specific classes are provided for software distribution, device configuration, and rest page management jobs. These classes perform any pre- or post-processing specific to the specific type of job, and they also allow all jobs of a specific type, such as software distribution, to be grouped together for convenience, such as for display in the Device Manager console.

Device Manager provides an object called deviceContext that can contain any Java object that is useful for maintaining device state information. When a device contacts Device Manager with a request, the DeviceCommunicationManager calls fire() to make the appropriate request. The fire() method calls a corresponding process() method so that the DMS servlet will process the event.

DeviceCommunicationManager creates the deviceContext object and passes it to the DMS servlet as part of the process() call. The DMS servlet passes the deviceContext object back to the DeviceCommunicationManager when it calls DeviceJob.doJob(deviceContext).

The communication between DeviceCommunicationManager and the DMS servlet is done through Java device management events. The DeviceCommunicationManager creates an event object of the appropriate type: DeviceConnectionEvent, DeviceJobProcessingCompleteEvent, DeviceRequestWorkEvent, and DeviceErrorEvent. DeviceManagementEvent is the abstract base class for device management events. When the DeviceCommunicationManager creates a device management event, it passes in the following information:

- The object source: The object creating the event, either the DeviceCommunicationManager or DeviceJob.
- The pervasive device ID.
- The deviceContext object.
- Other event-specific information that must be passed to the event as it is created. For example, on a DeviceRequestWorkEvent, it must pass the status indicating whether this is a new job request, or a request to continue a job.

In addition, DeviceErrorEvents must include the type of the error, and can optionally include a description of the error and a related Java exception error that is the underlying reason for the error.

Typical one-step job processing scenario

The following scenario describes the API calls and other communications between a device and Device Manager components during execution of a job that requires only one step to complete:

1. Through its plug-in, a device contacts DeviceCommunicationManager to request a job. The communications interface between the plug-in's device communication software and DeviceCommunicationManager is proprietary and determined by the plug-in developer.
2. DeviceCommunicationManager creates a DeviceRequestWorkEvent with a request of DEVICE_REQUEST_JOB, and then calls fire(DeviceRequestWorkEvent DEVICE_REQUEST_JOB). The event is constructed with information denoting the event source, the PervasiveDeviceID, and the DeviceContext objects.
3. The fire() method calls the associated process() method in the DeviceManagementServer class to process the event.
4. The DMS servlet finds the highest-priority job (called Job1Step in this scenario) for the device that is scheduled to be processed. It creates an object instance of the job, associates it with the device's PervasiveDeviceID, and initiates it with any associated job or device parameters.
5. The DMS servlet calls job1Step.performPreProcessing() to do any job-specific preprocessing.

6. The DMS servlet calls job1Step.doJob(), which is a method of the deviceJob class, to enable device-specific processing to be done. The implementation of the device job job1Step is unique to the device, and is written by the device vendor or integrator. Note that Job1Step was previously initialized with the jobParameters, the deviceParameters, and the PervasiveDeviceID, so this information does not have to be explicitly passed to the doJob() method. There are methods that a job can use to set and get a DeviceCommunicationManager job-specific object called jobContext. This object can be used as a holding place to store any job-related state information for jobs that require multiple interactions between the device and Device Manager.
7. The Job1Step object works with other plug-in components to return a message to the device instructing it to perform Job1Step.
8. The device performs Job1Step.
9. The device sends a message to DeviceCommunicationManager notifying it that the job has completed successfully.
10. The DeviceCommunicationManager creates a DeviceJobProcessingCompleteEvent with a status of JOB_COMPLETED. The event is constructed with information denoting the event source, the PervasiveDeviceID, and the deviceContext objects. The DeviceCommunicationManager calls the fire(DeviceJobProcessingCompleteEvent) method.
11. The fire(DeviceJobProcessingCompleteEvent) method calls the associated process() method on the DeviceManagementServer to process the event.
12. The DMS servlet calls job1Step.performPostProcessing() to perform any job-specific post-processing, including updating the device database with information about the software and the device.
13. The DMS servlet updates the Device Manager database with completion information.

The Javadoc for the device management server API provides details of all device management server API classes and their associated methods.

6.3.2.2 Creating a template file

For each new device class developed, a device class developer may provide a template file that describes the device-specific parameters required by the devices it manages. In the template file, the device developer describes each parameter, its syntax, and additional information to enable the Device

Manager console to build a user interface to present the device parameters to an administrator.

The format of the template file is common across all device classes. Device Manager provides a utility for installing template files when a particular device class is installed. The template file is optional, so if a plug-in writer is creating a plug-in for a device class that does not have any configurable parameters, a template file is not needed.

Template file format

A device class configurator included with Device Manager parses the template file and stores the device parameter definitions in the DEV_CLASS_TEMPLATE database table. The Device Manager console uses the device management API to retrieve the parameter definitions for a particular device class and to build a device configuration dialog. The dialog consists of at least one tab. Each tab includes device parameters, such as text fields or combo boxes. Device configuration values are stored in the DEVICE_PARM and DEVICE_CLASS_PARM tables and are passed to a job class when a device configuration job is processed for a device.

The template file is a set of keyword/value pairs that follow a bracketed tab or parameter name. The template file contains tab definitions and device parameter definitions. Device parameters are displayed in the user interface within a named tab. Both types of definitions consist of the tab or item name in brackets, followed by several keyword=value statements. Figure 82 shows an example of a template file.

```
[tab name]
type=Tab
keyword=value
keyword=value
:
:
[parameter name]
tab=tab name
keyword=value
keyword=value
:
:
```

Figure 82. Example of a device class template file

The name contained in brackets is looked up in the resource bundle. The name of the device class resource bundle must be DeviceClassTemplate.properties or DeviceClassTemplate_lang.properties, for example, \com\tivoli\dms\plugin\palm\PalmTemplate.properties. If the bundle does not exist, or the bracketed name is not found in the bundle, the label field is used as the displayed name on the user interface. The tab name is also used as the identifier when storing the parameter value in the DEVICE_PARM or DEVICE_CLASS_PARM tables. Keywords should conform to the list of predefined device parameter keywords.

Valid keyword/value pairs

Figure 27 lists the keywords and definitions for the template file. The keywords are valid only for parameters unless otherwise noted.

Table 27. Template file keywords

Keyword	Description
type	A value type, valid for tabs and as a parameter. This keyword is required for all tabs and parameters. Valid values are String, Int or Integer, Bool, or Tab.
range	Value range. This value is optional and is valid only for Integer types. It is specified in “start, end” format, where *start* is the starting value and *end* is the ending value. For example, range=1,10 means that the values can range from 1 to 10.
default	The recommended value. It is not stored as a value for any device or device class unless the administrator decides to take the default. This value is optional. If a range or group of choices is configured, the default must be within the range or one of the choices.
descriptionkey	A description of the parameter that should be displayed as flyover help for the item in the Device Manager console. The descriptionkey is used as a key in the resource bundle for the description text. If the resource bundle cannot be found, Device Manager uses the value specified for the description keyword as the flyover help text. If neither descriptionkey nor description is specified, no flyover help is available for that item in the Device Manager console.
description	A default description of the parameter, stored in the database. This keyword is optional, and is valid for both tabs and parameters. In the event that descriptionkey is not specified or the resource bundle cannot be found, the value specified for the description keyword is displayed in flyover help in the Device Manager console. If neither descriptionkey nor description is specified, no flyover help is available for that item in the Device Manager console.

Keyword	Description
labelkey	This keyword is optional and is valid for both tabs and parameters. The labelkey is used as a key in the device class resource bundle to retrieve the field name displayed for the Device Manager console. If the resource bundle does not exist, or the key is not found, the label is used instead as the displayed parameter name in the Device Manager console. The labelkey cannot contain any spaces. If a labelkey is not specified, the label keyword is used as the displayable field name for the Device Manager console.
label	The name of the parameter. This keyword is required and is valid for both tabs and parameters. The name in brackets is a string that is displayed as the field name for the Device Manager console. The label keyword is used only if a labelkey keyword is not specified.
choices	The list of possible values that are used in a combo box. This keyword is optional and only applies to String and Int parameters. Each item is separated by a comma. The user interface will always support a blank choice. For example, choices=A,B,C puts up a combo box with A, B, C, and blank as the possible choices. A list of choices can also be specified as choices=Value1|Label1,Value2|Label2,Value3|Label3.... The value specified on Value is stored in the database; the value specified on Label is displayed in the Device Manager console. For example, choices=1|Monday,2|Tuesday,3|Wednesday stores the values 1, 2, and 3 in the database, and displays Monday, Tuesday, and Wednesday in the Device Manager console. Special characters can be delineated using a backward slash (\) as an escape character. For example, to format an entry as 09,12,2000, you would specify it as 09\,12\,2000.
editable	Indicates whether the parameter value is editable. This keyword is optional. Valid values are T (for true) and F (for false).
tab	The tab to which the parameter belongs. This keyword is required. For example, tab=tcpip puts the item on the TCP/IP tab.
length	The maximum length of a String value, up to 1024 characters. This keyword is optional and is valid only for String types. Length cannot be specified if editable=false or if choices are specified.

Template file example

Figure 83 on page 186 shows and example if defining two tabs. The TCP/IP tab contains two text fields, and the Desktop tab contains one combo box and one text field.

```
# TCP/IP Tab Definition
#
[tcpip]
type=Tab
label=TCP/IP
description=TCP/IP settings
[pop3]
type=String
length=32
description=POP3 server address
label=POP3 Server:
labelkey=POP3
tab=tcpip
[ppp]
type=String
description=Dialup phone number
label=PPP Access Point:
labelkey=tcpipPPP
tab=tcpip
#
# Desktop Tab Definition
#
[desktop]
type=Tab
label=Desktop
description=Desktop settings
descriptionkey=Personalize your desktop settings
label=PPP Access Point:
labelkey=desktopPPP
[bgColor]
type=String
description=Background color of desktop
label=Background Color:
labelkey=bgcolor
choices=White,Black,Red,Yellow,Green,Blue
editable=true
tab=desktop
[timeout]
type=int
range=1,60
default=5
description=Default idle duration minutes to turn off screen
descriptionkey=Specify number of minutes for idle screen to wait before turning off
label=Idle screen Time-out:
labelkey=timeout
tab=desktop
```

Figure 83. Example of the TCP/IP tab and Desktop tab definitions

6.4 Device management API

The device management API provides the programming interface for managing devices, jobs, and related resources in the device management database. All Device Manager servers in a network share a common device management database. The device management API consists of a callable set of Java methods. These methods are called by many other entities, including:

- Other components of Device Manager. The Device Manager console uses the device management APIs to interact with the database to accomplish administrator-initiated operations, and the device management server API calls device management APIs to obtain information from the database that it needs to process jobs.
- Device management applications, such as the subscription management component of Tivoli Personalized Services Manager.
- External applications, such as billing systems, customer care, and self-care applications.

Using the device management API, you can manage the following types of data in the device management database:

- **Jobs**

  The device management API provides methods for creating, canceling, and deleting jobs and listing existing jobs. A job is a piece of work to be done for a device or group of devices, such as software distribution or configuration.

- **Device classes**

  Device class entries describe each of the device classes installed on a Device Manager server. The device management API provides methods for creating, updating, and deleting device class entries in the database, and for listing device classes.

- **Devices**

  Device entries describe a device that is owned by a subscriber. That is, the device's owner is assumed to have subscribed with the ISP for device management services. Device entries are created in the device management database when devices are first enrolled with an ISP. The device management API provides methods for creating, deleting, and updating device entries, for adding, deleting and updating the device parameters for devices, and for defining the list of software installed on devices. Additional methods provide the ability to filter the list of devices, based on search criteria.

- **Software**

  Software entries describe device software that has been packaged for distribution to devices. Each software entry is associated with devices of a specific device class. That is, if a particular software application runs on devices supported by different plug-ins, you must define entries for each software-device class relationship. The device management API provides methods for creating, deleting, and updating software entries in the database. An additional method provides the capability to list software based on matching specific software attributes.

- **Device Manager servers**

  Device Manager server entries describe each of the Device Manager servers in the network. The device management API provides methods for creating, updating, and deleting Device Manager server entries in the device management database, and for listing known servers. Note that Device Manager itself manages the entries in these tables by registering itself automatically when it is started; the programmer does not need to update these entries.

6.4.1 Managing devices

The device management API provides a set of callable methods for managing devices and their related resources in the device management database. The database contains entries describing devices, device software, related parameters, and Device Manager servers. Methods are provided for creating, deleting, and updating these entries, as well as for searching the database for entries matching certain criteria.

6.4.1.1 Device management business objects

The device management API provides the following business objects, which are based on the TxObject class in Tivoli Personalized Services Manager's Integration Toolkit. The device management business objects are:

- **DMDevice** is used to create, delete, and update devices and device attributes in the DEVICE table. Additionally, you use DMDevice to list devices and their related attributes, such as device parameters or software available for install, from the DEVICE_PARM or INSTALLED_SW database tables. For example, you would use this object to query the database for a list of software installed on a particular device.
- **DMDeviceClass** is used to create, delete, and update device class entries and parameters in the associated DEVICE_CLASS table, as well as to list device classes and their related attributes (such as device class parameters) from the DEVICE_CLASS_PARM table.
- **DMServer** is used to create, delete, update, and list Device Manager server entries in the associated DMS_SERVER table.
- **DMSoftware** is used to create, delete, and update software information in the SOFTWARE table, as well as to list software packages and device

class data. Note that since this business object does not know about the INSTALLED_SW table, you need to use the DMDevice business object to list software that has actually been installed as opposed to software that is available for installation.

- **DMJobClass** is used to create, delete, update, and list information in the JOB_CLASS_TABLE table about job classes and their relationship to device classes.

Each device management business object provides two levels of APIs to manipulate database entries:

- The low-level APIs require the caller to get an instance of the business object, call the various set methods to set the required fields, get an integration toolkit Transaction instance, and pass the business object to the desired Transaction method.
- The high-level APIs are built on top of the low-level APIs. These higher-level APIs manipulate Transaction objects and make function calls as needed, and they automatically join with additional database tables (if appropriate.)

For most programming purposes, the high-level APIs are the preferred programming method. A programmer who is very familiar with Tivoli Personalized Services Manager's integration toolkit may choose to use the low-level APIs in certain situations.

6.4.1.2 Device Manager database tables

The Device Manager database consists of a series of relational database tables for storing information about devices and their associated resources. The tables you can use are:

- **DEVICE**: The device table maintains a base set of information for each device being managed. Each device is associated with a single owner; for example, a Tivoli Personalized Services Manager subscriber.
- **DEVICE_CLASS**: A device class represents a type of device supported by Device Manager, for example, all devices that use PalmOS. For each device class, Device Manager is configured with the device plug-in, which consists of a Java class that provides the interface to the device and a template file that describes the set of device parameters required by devices of that device class.
- **DEVICE_PARM**: A device parameter is a key/value pair that provides configuration information to a device, such as IP Gateway or DNS Server, which the device uses. This table maintains the parameters set for specific devices.

- **DEVICE_CLASS_PARM**: This table maintains the device parameters set or all devices of a particular device class.
- **DEVICE_CLASS_TEMPLATE**: A device class template file is provided with each device class defined to the Device Manager and describes the set of device parameters required by devices of that device class.
- **SOFTWARE**: This table maintains information about software that can be downloaded to specific devices.
- **INSTALLED_SW**: This table maintains information about what software has been downloaded to specific devices.
- **DMS_SERVER**: This table maintains the list of Device Manager servers in the network. Each Device Manager server coordinates the execution of device management jobs scheduled for devices. Scheduled jobs are maintained in a central database.
- **JOB_CLASS_TABLE**: A job class is a Java class that provides the implementation of a high-level device management operation, such as Software Distribution, Device Configuration, or Rest Page Management. This table maintains the set of job classes defined to the Device Manager and the set of device classes that each of these job classes supports.

For more information about table entries, refer to Chapter 5, “Device Manager database” on page 145.

6.4.2 Managing jobs

The device management API provides a set of classes for manipulating the information about jobs in the device management database. Methods are provided to submit, cancel, and delete jobs, as well as to query the current state of jobs.

6.4.2.1 Job management business objects

The device management API uses two business objects for job management. These business objects are based on the business objects defined for the Tivoli Personalized Services Manager Integration Toolkit.

- **DMSSubmittedJob** is the business object representing a submitted job. The API uses the base Integration Toolkit business object's methods for creating, updating, reading, and deleting the DMSubmittedJob object.
- **DMChildJob** is the business object representing a child job. Child jobs cannot be created, deleted, or updated through the Device Manager console. The child job object is used primarily for checking on the status of a job.

6.4.2.2 Job management database tables

The job management database consists of a series of relational database tables for storing information about jobs submitted on a specific Device Manager. The tables you can use are:

- **SUBMITTED_JOB**: This table contains information about jobs submitted for a specific device, all devices of a specific device class, and all devices.
- **ACTIVE_JOB_HISTORY**: This table contains IDs for jobs that have completed on specific devices. It is used to record when a job completes successfully on a device so that the job is not executed again on the same device. Unsuccessful completions are also recorded as an aid to problem determination. Records are removed from the table when the job is deleted from the SUBMITTED_JOB table.
- **JOB_PARM**: This table contains device parameters provided when the job is submitted.

6.4.2.3 Displaying information in the DM console

The Device Manager console administrator has the capability to limit how many items are returned on a query. To support this, the DMResult class has two methods:

- **isMore()** returns true if more than the maximum number of items would be returned on the query specified in the query. It also causes the warning message shown in Figure 84 to be displayed.

```
DYM0115W The number of items matching the filter criteria exceeded the
specified maximum of maximum_number.  Only maximum_number items
were returned.
```

Figure 84. Example of isMore() method

- **getObjects()** returns a vector of the following “lightweight” (scaled-down) objects:
  - DMDeviceData
  - DMSubmittedJobData
  - DMChildJobData
  - DMSoftwareData
  - DMServerData
  - DMDeviceClassData
  - DMJobClassData

Chapter 7. Using the DM functions

This chapter explains how to use the Device Manager functions, software distribution, and device configuration. It describes the operational tasks, step by step, and then shows the operations for both the Device Manager server and subscriber using sample scenarios.

Before you can start creating jobs, you need to have administrator privileges to manage target devices through the Device Manager console.

For information on how to install and customize the Device Manager console, refer to the online manual *Tivoli Personalized Services Manager Device Manager: Administration*.

7.1 Device agent installation

To use any Device Manager function, subscribers need to install the device agent program on their pervasive devices. This is developed together with each device plug-in. The subscribers connect to the Device Manager server by invoking this agent program.

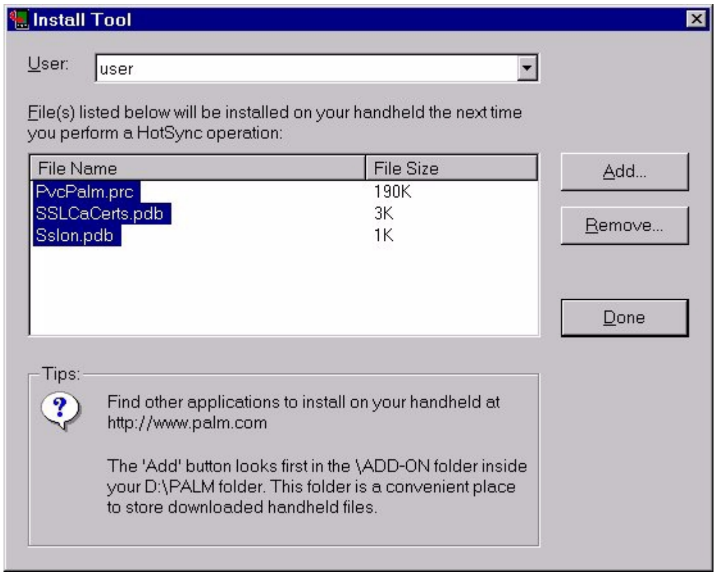
In this section, we describe how to install and setup the device agent program for the Palm device. For another device agent program installation, refer to the online manual, *Tivoli Internet and Personalized Services Manager Device Manager: Device Plug-in Notes*.

7.1.1 Installing the device agent program

First, you need to install the device agent program for the Palm device. This should be done using the HotSync operation. Complete the following tasks:

1. Connect the Palm cradle to the HotSync-installed PC, and place the Palm device in the cradle.
2. Open the Install Tool window, shown in Figure 85 on page 194, of the Palm Desktop. Then, install the PvcPalm.prc file.

   If you need to use SSL to communicate with the Device Manager server, you need to install the certificate file and set the agent to SSL-enabled. This is done by installing the PDB file to perform the setting or by using a device configuration job. PDB files contain database records that are used by the Palm OS to store the application data, and they have features unique to the Palm OS.



User: user

File(s) listed below will be installed on your handheld the next time you perform a HotSync operation:

File Name	File Size
PvcPalm.prc	190K
SSLCaCerts.pdb	3K
Sslon.pdb	1K

Find other applications to install on your handheld at http://www.palm.com

The 'Add' button looks first in the \ADD-ON folder inside your D:\PALM folder. This folder is a convenient place to store downloaded handheld files.

Figure 85. Install Tool window

3. The DevAgent icon, shown in Figure 86, appears on the Palm display when installation process completed.

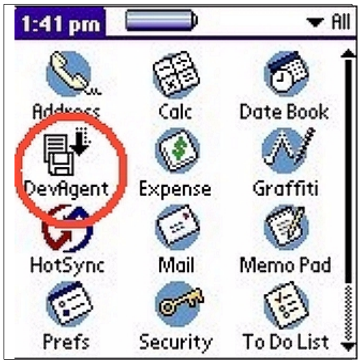


Figure 86. DevAgent icon

7.1.2 Device settings

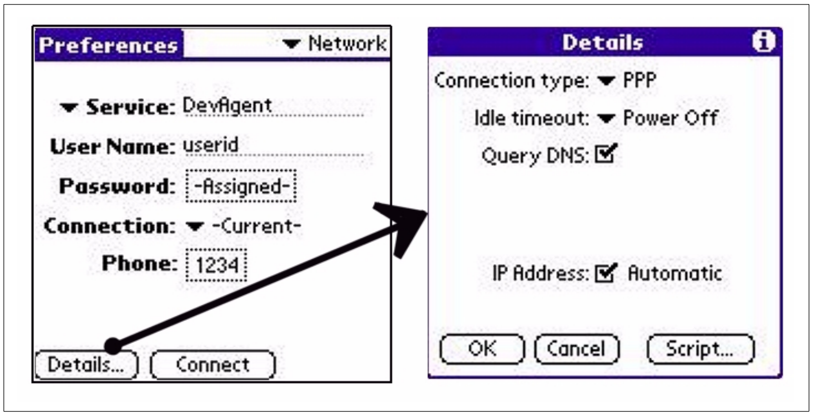
After installing the device agent, you should configure the network settings of the Palm, and set up the agent.

You need some information from the service provider when you try to set up your device. The information you need includes:

- PPP userID
- PPP password
- Network service name
- ISP’s phone number
- ISP’s userID (TPSM subscriber userID)
- ISP’s password (TPSM subscriber password)
- The address and port number of Device Manager server
- Servlet name
- Proxy address (if necessary)

Following are the step-by-step instructions for setting up the Palm device.

1. Set the Palm device network parameters and connection preferences from the **Prefs** icon. Figure 87 shows the Preferences and Details windows.
   a. From the **Connection** menu, select the proper modem, and set the value.
   b. From the **Network** menu, select **create new service**, and set the ppp userID/password, connection, and phone number.
   c. Click **Details**, and set the DNS server address.



Preferences ▼ Network

▼ Service: DevAgent

User Name: userid

Password: -Assigned-

Connection: ▼ -Current-

Phone: 1234

Details

Connection type: ▼ PPP

Idle timeout: ▼ Power Off

Query DNS: ☑

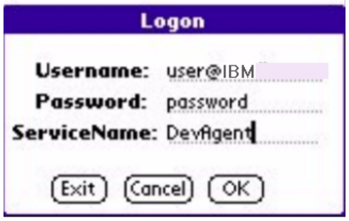
IP Address: ☑ Automatic

Figure 87. Network setting from Prefs icon

2. Click **Home** to return to the main menu. Next, click the **DevAgent** icon. Since this is the first time the device agent program is invoked, additional setting windows appear.
3. In the confirmation window, respond with **Never DSP**. When the Logon window, shown in Figure 88, is displayed, enter the subscriber information in the following format:

- **Username**: username@realmname
- **Password** TPSM user's password
- **ServiceName** The service name of the ISP

This information should be provided by the ISP before you try to install device agent program.



Logon

Username: user@IBM

Password: password

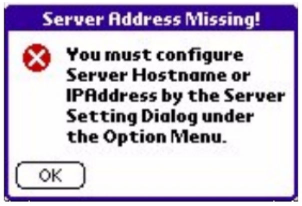
ServiceName: DevAgent

Figure 88. Logon setting window

Logon window

The Logon window only appears the first time the device agent is invoked. If the wrong information is set, you must reinstall the device agent program to correct it.

4. When the warning window shown in Figure 89 is displayed, click **OK**.



Server Address Missing!

You must configure Server Hostname or IPAddress by the Server Setting Dialog under the Option Menu.

Figure 89. Warning window

5. Next, the PalmAgent window, shown in Figure 90, is displayed.

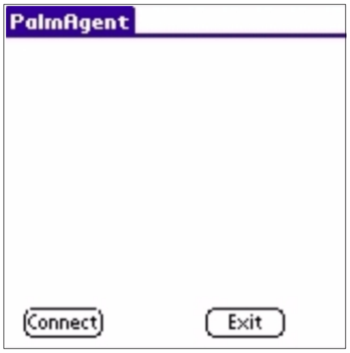


Figure 90. Device agent main window

6. Click the **List** menu icon, and open **Options** -> **Server setting** as shown in Figure 91.

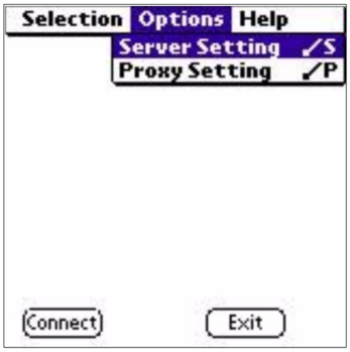
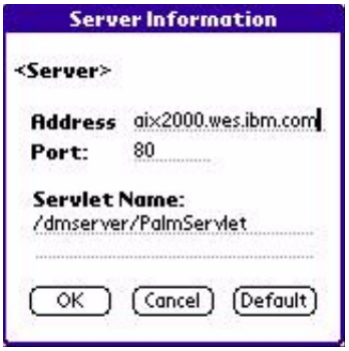


Figure 91. Select Server Setting option

7. Enter the server address, server port, and the servlet name in the **Server Information** window, shown in Figure 92 on page 198, and then click **OK**.


Server Information

<Server>

Address aix2000.wes.ibm.com

Port: 80

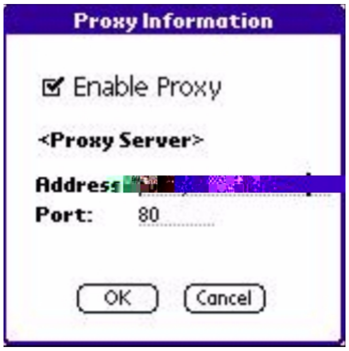
Servlet Name:
/dmserver/PalmServlet

Figure 92. Server Setting

By clicking the **Default** button, you can set **Port** and **Servlet Name** fields to default. The value to be set to the Port field depends on the SSL setting of the agent. In cases where SSL is enabled, the Port field is set to 443, and in cases where SSL is disabled, the Port field is set to 80. The Default button always sets the value of the Servlet Name field to /dmserver/PalmServlet.

For instructions on how to enable SSL, refer to 4.4, “Enabling SSL” on page 141.

8. If your ISP instructs you to set the proxy address, select the **Proxy Setting** from the **Options** menu, shown in Figure 93. Then, mark the square to the left of **Enable Proxy**, and set the proxy server address in the Proxy Information window, shown in Figure 94 on page 200. To apply your settings, click **OK**.


Proxy Information

☑ Enable Proxy

<Proxy Server>

Address [redacted]

Port: 80

Figure 93. Proxy Setting

Now you can connect to the Device Manager server by clicking **Connect** in the **PalmAgent** window, or by clicking the **DevAgent** icon.

7.1.3 Device setting with Palm Cradle

The Palm device can communicate using a modem. Using Device Manager's built-in Palm device plug-in support, the Palm device can also communicate using a cradle. To support this connection, the Palm plug-in provides the conduit software named TPSM 1.1 Palm Cradle Support for Windows.

The installation program named /usr/lpp/TivDMS/agents/palm/CondInst.exe is shipped with Device Manager. This contains a plug-in to the Palm Desktop software, named CondAgent.

The conduit PC needs to have Palm Desktop software installed and be able to resolve the host name of the DM servers properly.

To install TPSM 1.1 Palm Cradle Support for Windows, simply run the CondInst.exe program on the machine with Palm Desktop installed. It will temporarily stop the Palm HotSync Manager, install the cradle support software, and then restart the Palm HotSync Manager.

To connect to the Device Manager from the cradle-connected Palm device, simply click the **DevAgent** icon as you connect the DM server using a modem. The settings are the same as those used for modem communication.

The device agent has a function to distinguish whether the device is connected to the modem or the cradle.

Note

Some types of Palm devices or Palm modems cannot be distinguished by the device agent. In that case, the agent displays a window asking whether the device is connected using a cradle or modem.

After the agent notices that the device is connected to the cradle, it invokes the HotSync program. TPSM 1.1 Palm Cradle Support for Windows is invoked as a plug-in to HotSync and connects to the Device Manager server as if it is a device agent. It creates a folder on the local disk, with the same name as the device ID, and stores the configuration parameters and the distributed software in that folder.

7.2 Software distribution

Software distribution is one of the important functions of the Device Manager. The service provider can centrally manage software that is installed in a subscriber's personal device using this function. In this section, we describe how to plan and prepare for tasks that you need to perform when you use the software distribution function for the Palm device.

Although the preparation tasks are almost the same as when you use other types of devices, like Windows CE or Aero 8000, supported parameters are slightly different. For detailed information about the software distribution function for each device class, refer to the online manual *Tivoli Internet and Personalized Services Manager Device Manager: Device Plug-in Notes.*

7.2.1 Planning for software distribution

The system administrator uses the Device Manager console or an application to create a software distribution job and submit the job for processing.

7.2.1.1 Understanding the flow

You need to know the procedure to prepare files and create a software distribution job. We describe each task in the following sections.

Figure 94 shows the flow for distributing software to pervasive devices.

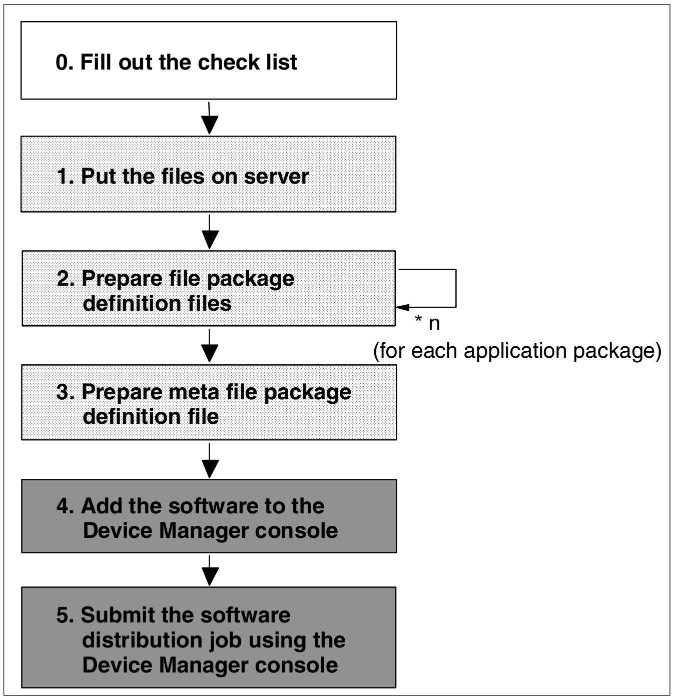

0. Fill out the check list
1. Put the files on server
2. Prepare file package definition files
* n
(for each application package)
3. Prepare meta file package definition file
4. Add the software to the Device Manager console
5. Submit the software distribution job using the Device Manager console

Figure 94. Software distribution preparation flow

7.2.1.2 Making the checklist

You need to know some items needed to make and characterize a software distribution job, for example, whether it overwrites existing software or not. Or, let the subscriber choose whether to install the software now or later.

We recommend that you make a checklist like the one shown in Table 28 as your first step. This table contains only the minimum items needed to create a software distribution job. You may have to add items based on your system needs.

Table 28. Distribution check list

Item	Description	Valid value
Target device class	Target device you want to distribute	- Aero8000 - Iad - Palm - Wince
Target device	Target device you want to distribute	- All devices of a device class - A single device
Target realm	Target realm name	existing realm(s)
Target deal	Target deal name	existing deal(s)
Activation date	Date job become in service	YYYY MM/DD hh:mm
Expiration date	Date job become out of service	YYYY MM/DD hh:mm
Software name	Name for software package	Any value
UserSelection	User selection for this package For detail, refer to Table 30 on page 207	- no - yes/delay - yes/reject - yes/delay/reject
Application name	Name for each application in this software package	Any value
Application size	The needed space in bytes	Any valid numeric value for example, 1000
Installed software check	Determines if a file is overwritten on the target device	- y=do not overwrite - n=overwrite - v=version check

7.2.2 Placing the files on the server

You need to put the files to be distributed on the server machine. The location must be accessible using the FILE or HTTP protocol. The server administrator can identify an appropriate directory.

As long as the files are accessible using the FILE or HTTP protocol, an administrator can put all the files where he wants, even in the same directory. He can also name a file package definition file, and a meta file package definition file.

We recommend that you decide on your naming rules first; for example, applicationname.def for a file package definition file, and packagename.meta for a meta file package definition file. A set of rules for your directory structure is also needed. Figure 95 shows a sample directory structure.

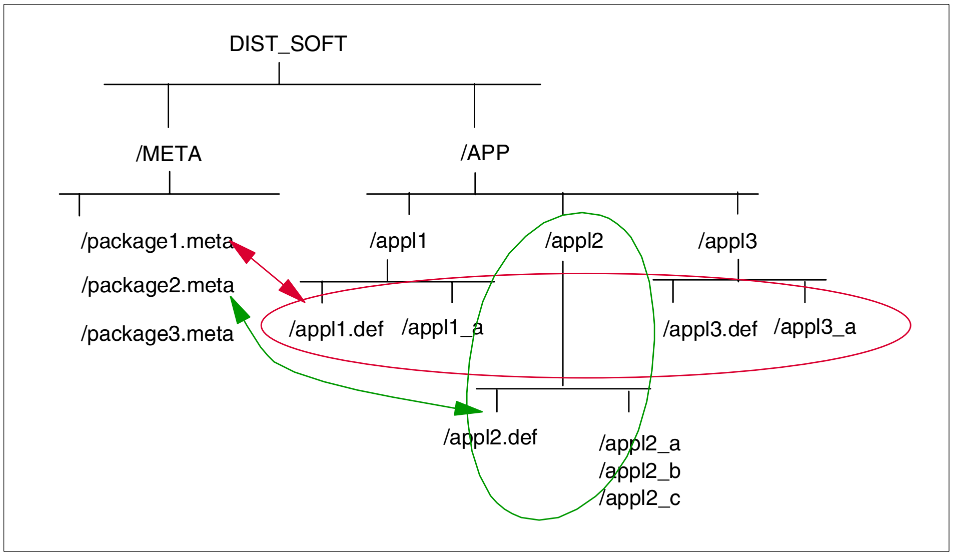


Figure 95. Sample directory structure e for software distribution

7.2.3 Preparing the file package definition file

The file package definition file is comprised of a header and five sections. The sections are:

- Keyword options
- Files and directories to be distributed
- Nested file packages (not used for DMS 1.1 device plug-ins)
- Excluded files (not used for Palm)
- Extra (not used for Palm)

A file package definition file must have four percent symbols (%%%%) delimiting the five parameter sections. Even if there are no parameters for a section, there must be a percent symbol delimiting each section. There is no delimiter between the header and the first section. The first section starts on the second line of the file package definition file.

The valid options for each section depends on the device plug-ins. In this section, we describe the file contents for a Palm device.

For information about the valid options for each device class, refer to the online manual *Tivoli Internet and Personalized Services Manager Device Manager: Device Plug-in Notes*

Figure 96 shows a sample file package definition file for a Palm device.

```
aix2000[/S_DIST/APP/pscape]#cat pscape.def
#*DFP-v1.00 DMS FilePack (version 1.0)
need_space=289000
no_overwrite=y

%
Palmscape35.prc Xd="Palmscape for Palm OS3.5" s=/S_DIR/APP/pscape

%
%
%
aix2000[/S_DIST/APP/pscape]#
```

Figure 96. Sample file package definition file for Palm device

Header section

The header line must be the first line of the file package definition file. The header typically lists the version number, for example:

```
#*DFP-v1.00 DM FilePack (version 1.00)
```

Keyword options section

The keyword options section contains a list of keywords and values which control the behavior of the file distribution job. The format for the keyword section is a keyword followed by an equal sign (=) and then a value for that keyword. Space around the equal sign is not allowed.

Table 29 shows a list of keywords supported for Palm devices.

Table 29. Keyword options for Palm

Keyword	Description	Valid values
need_space	Specifies the needed space in bytes	Any valid numeric
no_overwrite	Determines if a file is already installed on the target	- y=do not overwrite - n=overwrite(default) - v=version check
src_relpath	Specifies the path on the source host relatives to the files and directories in the file package	A full path or a relative path from the current directory on the source host

Free space requirement

For software distribution, the amount of free space available on the device should be at least twice the size of the software distribution job plus the agent heap space.

For example, if an application package requires 100 KB, then the amount of free space needed on the device would be (100*2) + (32 to 48)=232 to 248 KB. If you attempt to install software when there is not sufficient space, the following error message appears:

```
There is not enough storage on this device for this job
```

Files and directories section

This section contains the names of the files that comprise the application package.

Each file entry is listed on a separate line. For a file entry, you can optionally provide a file description, or override the source path previously specified with the src_relpath keyword, or both. The options are:

- **s**: Specifies a path on the source host from which to obtain the file and directory. Overrides the src_relpath keyword option.
- **Xd**: Specifies the file description. The format is Xd="description".

Nested file packages section

The nested file package section is not supported in DMS 1.1 and is reserved for future use. Even though this section is not supported, you must include a % symbol to delimit this section.

Excluded files section

The exclude files section is not supported in DMS 1.1 and is reserved for future use. Even though this section is not supported, you must include a % symbol to delimit this section.

Extra section

The extra section is not supported in DMS 1.1 and is reserved for future use. Even though this section is not supported, you must include a % symbol to delimit this section.

7.2.4 Preparing the meta file package definition file

The meta file package definition file provides software distribution properties for the job; for example, to allow the user to delay or to reject the software distribution job. These properties are called package properties.

The meta file package definition file also has a numbered application stanza, [Application#], for each application package in the software distribution job. The application stanza must be uniquely named and each stanza must begin with the word Application. The application stanza identifies application properties. The file package definition file is identified with one of the application properties, the ApplicationURL keyword. There are additional properties identified for each application package, such as a version and description.

The meta file package definition file contains:

- **Package properties** - Define package characteristics
- **Application properties** - Define each application characteristics

Figure 97 on page 206 shows a sample meta file package definition file for a Palm device.

```
aix2000[/S_DIST/META]#cat game.meta
PackageUserSelection=no
PackageName=WebBrowser
PackageVersion=3.5
PackageDescription=WebBrowser Package for PalmOS 3.5

[Application1]
ApplicationUrl=APP/pscape/pscape.def
ApplicationSelectionDisable=no
ApplicationName=Palmscape
ApplicationVersion=3.5
ApplicationDescription=Palmscape3.0 for PalmOS3.5
aix2000[/S_DIST/META]#
```

Figure 97. Sample meta file package definition file

For each software distribution job, there is one meta file package definition file. When an administrator adds software to the Device Manager console, the meta file package definition file is named in the Software URL field.

Refer to 7.3.3, “Server operations” on page 209, for details on adding software and submitting jobs for software distribution.

Package properties section

The package properties section contains a list of keywords and values which describe and control the behavior of the software distribution job. The format for the package properties section is a keyword followed by an equal sign (=) and then a value for that keyword. Space around the equal sign is not allowed.

The setting for the PackageUserSelection keyword allows the user to choose to download the software distribution job now, delay the download, or reject the software distribution job. The package properties also provide a name, version, and description.

Table 30 provides a list of package properties for the Palm device.

Table 30. Package properties for the Palm device

Keyword	Description	Valid value
PackageUserSelection	User selection for this package	- no - yes/delay - yes/reject - yes/delay/reject
	no: no choice by user, always install	
	yes/delay: user can choose to install now or delay choice	
	yes/reject: user can choose to install now or cancel installation	
	yes/delay/reject: user can choose to install now, delay choice, or cancel installation	
PackageName	Name for the package	Any value
PackageVersion	Version for the package	Any value
PackageDescription	Description for the package	Any value

Application properties section

The application properties section contains a list of keywords and values which describe and control the behavior of each application package in the software distribution job.

The meta file package definition file has a separate application stanza for each application package like a sample file shown in Figure 97.

Within each application stanza, the ApplicationURL keyword identifies the file package definition file for the application package. The application stanza also has properties that are specified for each application package, such as version and description.

Table 31. Application properties for the Palm device

Keyword	Description	Valid values
ApplicationURL	Location the file or URL that contains the file package definition file	- A valid URL - file name (relative path or full path)
ApplicationSelectionDisable	Determine user’s choice	- yes=application is always installed - no=application is only installed if the user checks it

Keyword	Description	Valid values
ApplicationName	Name for the application package	Any value
ApplicationVersion	Version for the application package	Any value
ApplicationDescription	Description for the application package	Any value

7.3 Sample scenario for software distribution

Following Figure 94 on page 200, this section explains, step-by-step, how to create a software distribution job for the Palm device.

7.3.1 Scenario 1

An ISP advertises for subscribers. This ISP allows Palm III and V users to be served. A user subscribes to the ISP using the Web or a phone call, and then he or she is granted the right to download the Device agent. Together with the Device agent, the subscriber receives the information necessary to connect to the ISP, and to the Device Manager server.

The subscriber purchases the Palm device on the market, and sets up the device agent. The subscriber also wants some network settings to be set at the first connection. These network settings are common to all devices.

The ISP assumes that the Palm device of the new subscriber does not have a Web browser installed, so at the first connection, the ISP wants to install a Web browser to the subscriber’s Palm device. The ISP also gives free game software to their new customers if the customers want to install it.

7.3.2 System environment

Figure 98 shows the system environment we used for this scenario. We used two UNIX servers, one IBM RS/6000 acting as the Device Manager server and the other is a Sun Enterprise server acting as a Services Manager server. The Device Manager database is installed in the Sun Enterprise server machine.

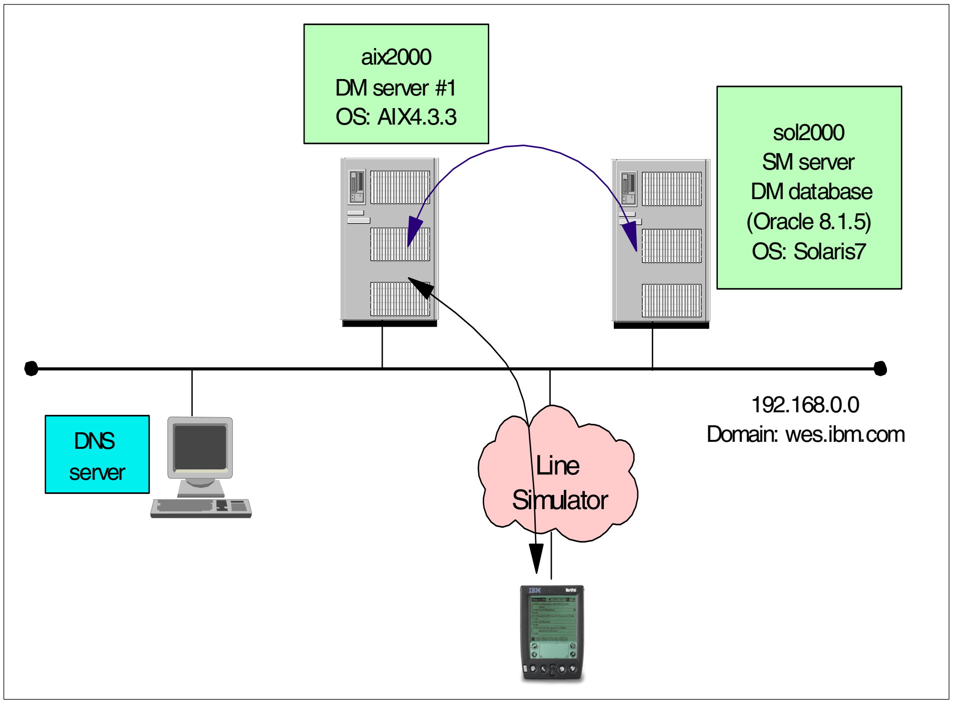


Figure 98. Sample scenario system environment

7.3.3 Server operations

According to the scenario mentioned in 7.3.1, “Scenario 1” on page 208, the system administrator needs to prepare the following two jobs:

- **Job1**: **Distribute browser named PalmScape**
  This software is automatically installed when a new device connects to the Device Manager server the first time.
- **Job2**: **Distribute packaged software**
  The subscriber can choose to install the packaged software now or later, and can select which software packages to install.

The following sections describe the tasks that the system administrator needs to perform to distribute software for the Palm device.

7.3.3.1 Filling out the checklist

We completed the checklist shown in Table 32 for our sample scenario.

Table 32. Sample scenario job checklist

Item	Job1	Job2		
Target device class	Palm	Palm		
Target device	All devices	All devices		
Target realm	IBM	IBM		
Target deal	PACK1	PACK1		
Activation date	2000 11/01 14:00	2000 11/01 00:00		
Expiration date	2002 12/01 12:00	2002 01/01 00:00		
Software name	WebBrowser	GamePackage		
UserSelection	no	yes/delay		
Application name	Palmscape	BOMBRUN	pocketra	Astroids
Application size	289000	6000	15000	65000
No_overwrite	y	v	v	v

7.3.3.2 Putting the files on the server

For this scenario, we decided to use the directory structures shown in Figure 99. We put the files to be distributed, along with the definition files, on the server and made sure the files had read authority.

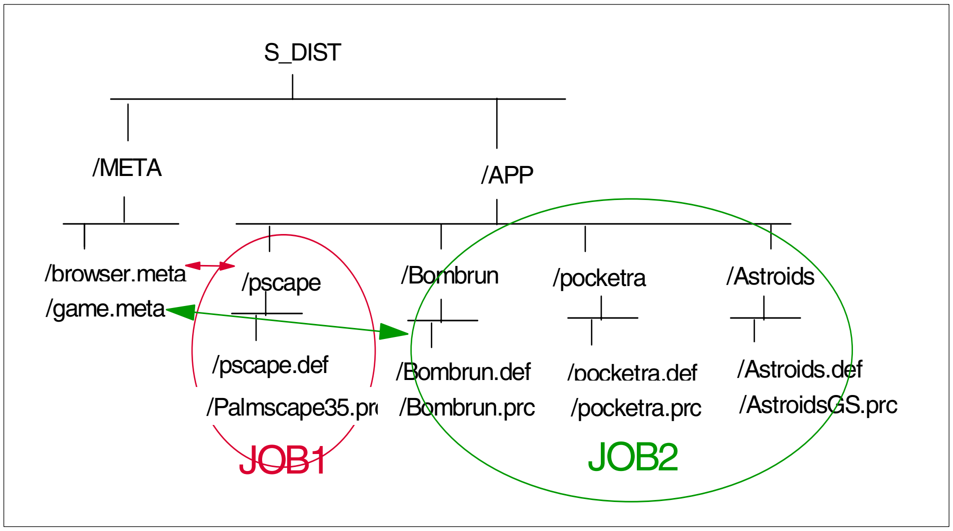


Figure 99. Sample scenario Directory Structure

7.3.3.3 Preparing the file package definition file

Based on the checklist, we prepared the file package definition file for each of the four application packages. Figure 100 is a sample file package definition file for an application package named Asteroids.

```
aix2000[/S_DIST/APP/Astroids]#cat Astroids.def
#*DFP-v1.00 DMS FilePack (version 1.0)
need_space=65000
no_overwrite=v

%
astroidsGS.prc Xd="astroidsGS Palm file"

%
%
%

aix2000[/S_DIST/APP/Astroids]
```

Figure 100. Sample file package definition file for Scenario 1

7.3.3.4 Preparing the meta file package definition file

Based on the checklist, we prepared the meta file package definition files for each job; that is, we prepared two meta file package definition files. Figure 101 on page 212, shows a sample meta file package definition file for Job2 named game.meta.

```
aix2000[/S_DIST/META]#cat game.meta
PackageUserSelection=yes/delay
PackageName=GamePackage
PackageVersion=1.0
PackageDescription=Game Applications

 [Application1]
ApplicationUrl=APP/Bombrun/Bombrun.def
ApplicationSelectionDisable=no
ApplicationName=BOMBRUN
ApplicationVersion=1.0
ApplicationDescription=BOMBRUN

 [Application2]
ApplicationUrl=APP/pocketra/pocketra.def
ApplicationSelectionDisable=no
ApplicationName=pocketra
ApplicationVersion=0.50
ApplicationDescription=pocketra

 [Application3]
ApplicationUrl=APP/Astroids/Astroids.def
ApplicationSelectionDisable=no
ApplicationName=Astroids
ApplicationVersion=1.52
ApplicationDescription=Astroids
aix2000[/S_DIST/META]#
```

Figure 101. Sample meta file package definition file for Scenario 1

At this point, you are ready to add the software and submit jobs for software distribution using Device Manager console.

7.3.3.5 Adding the software to the Device Manager console

To add the new software to the Device Manager console, follow these steps:

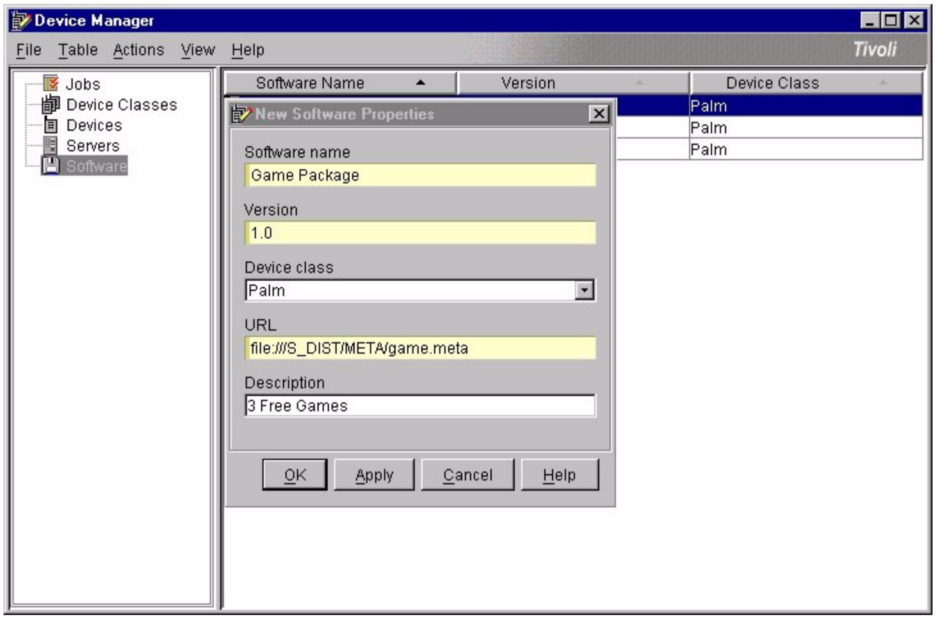
1. Right-click **software** and select **New Software** from the context menu. The New Software Properties window, shown in Figure 102, is displayed.
2. Complete all the required fields, including the URL, which is the software URL. Specify the meta file package definition file name using the HTTP or FILE protocol.

   To prevent duplication, no two software packages can have duplicate information in all three of the following fields:

   - Software name
   - Version
   - Device class

   At least one of these fields must be different for any two software packages.

3. To save this information and close the window, click **OK**.



Software Name	Version	Device Class
		Palm
		Palm
		Palm

New Software Properties

Software name: Game Package

Version: 1.0

Device class: Palm

URL: file:///S_DIST/META/game.meta

Description: 3 Free Games

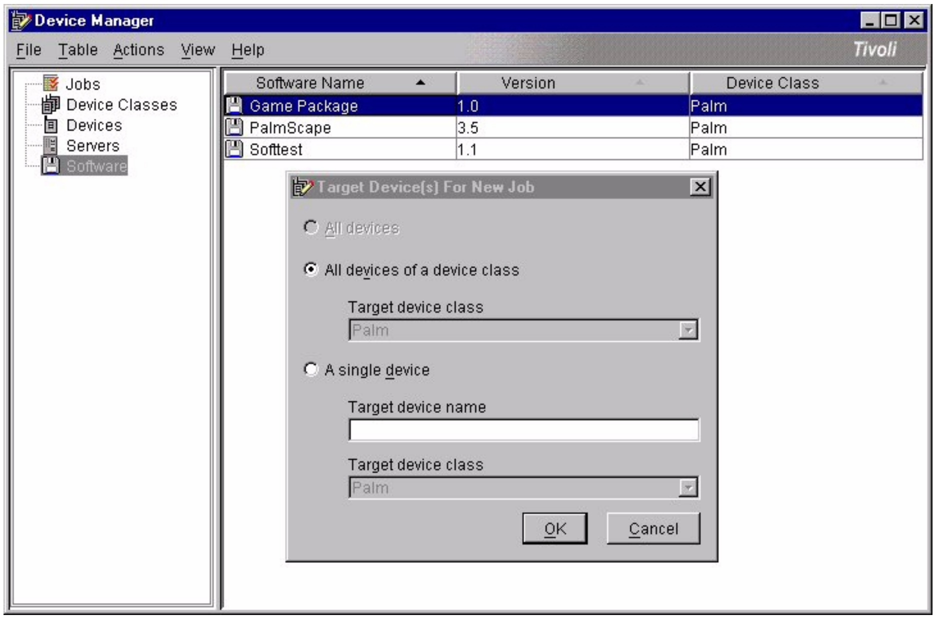
Figure 102. New Software Properties window

7.3.3.6 Submitting the software distribution job

Before submitting a job to the customer’s device, you have to submit a test job to a specific test device. This is required to ensure that the package has been created properly and its associated software has been properly deployed to the customer’s environment.

It also ensures that the software definition, specifically the URL referencing the software package, has been properly configured in Device Manager.

1. Make sure that a software package that was added in the previous operation, is highlighted in the right pane of the console. Right-click it and select **Submit Job** from the context menu. The Target Device(s) For New Job, shown in Figure 103 on page 214, is displayed.
2. Select **All devices of a device class**, and make sure **Palm** is selected in Target device class.
3. To save this information and close the window, click **OK**. The New Job Properties window is displayed.



Software Name	Version	Device Class
Game Package	1.0	Palm
PalmScape	3.5	Palm
Softtest	1.1	Palm

Target Device(s) For New Job

○ All devices

● All devices of a device class

Target device class: Palm

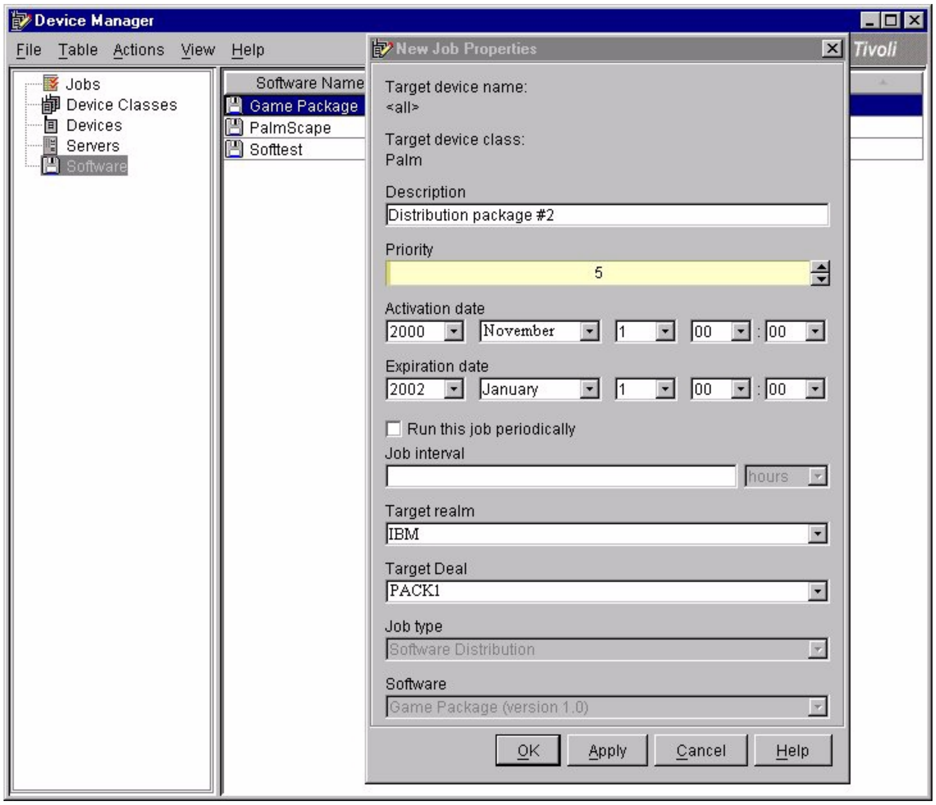
○ A single device

Target device name:

Target device class: Palm

Figure 103. Target Device For New Job window

4. Make the appropriate changes to the fields in the **New Job Properties** window, shown in Figure 104. Fill in the Activation date, Expiration date, Target realm, and Target Deal fields using the information from the checklist in Table 32 on page 210.
5. Click **OK** to submit the job.



Target device name:
<all>

Target device class:
Palm

Description
Distribution package #2

Priority
5

Activation date
2000 November 1 00 : 00

Expiration date
2002 January 1 00 : 00

□ Run this job periodically

Job interval
hours

Target realm
IBM

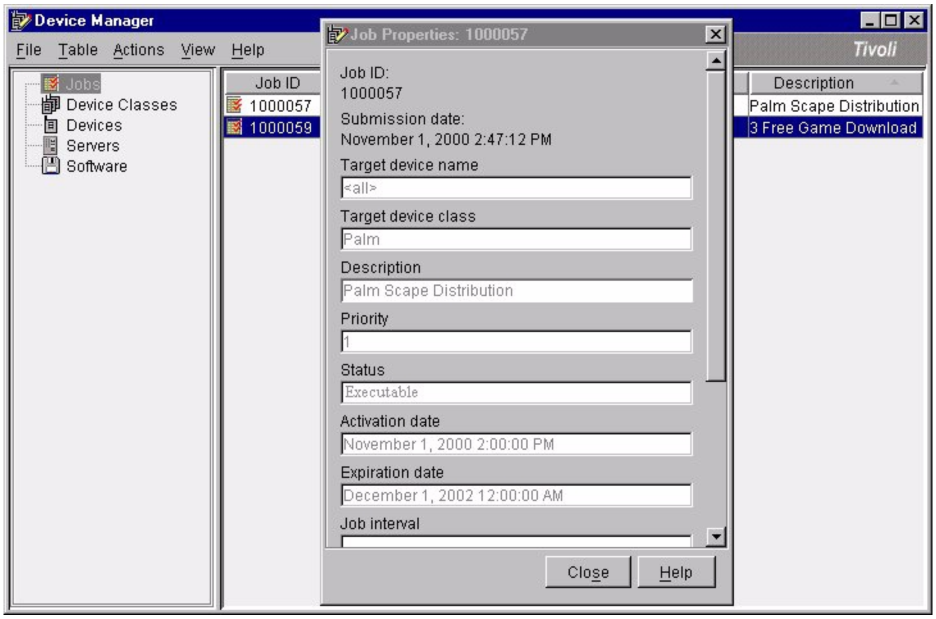
Target Deal
PACK1

Job type
Software Distribution

Software
Game Package (version 1.0)

Figure 104. New Job Properties window

6. Right-click a software package that was previously submitted and select **View Applicable Jobs** from the context menu. A secondary window, shown in Figure 105 on page 216, is displayed showing all software distribution jobs applicable to the software.
7. Right-click a job in the right pane of the console and select **Properties** from the context menu. The Job Properties window is displayed.
8. Click **Close** to close the window.



Job ID	Description
1000057	Palm Scape Distribution
1000059	3 Free Game Download

Job ID:
1000057

Submission date:
November 1, 2000 2:47:12 PM

Target device name
<all>

Target device class
Palm

Description
Palm Scape Distribution

Priority
1

Status
Executable

Activation date
November 1, 2000 2:00:00 PM

Expiration date
December 1, 2002 12:00:00 AM

Job interval

Figure 105. Job Properties window

You cannot modify the fields in the Job Properties window because you have already submitted this job. If there are any fields you want to change, you need to cancel the job and submit it as a new job. When all fields are correct, the administrator operations are complete.

7.3.4 Client operations

Now that the Palm device has been set, connect to the Device Manager server. This is the first time a connection to the server has been made, so this device should be enrolled to the Device Manager database. According to the scenario mentioned in 7.3.1, “Scenario 1” on page 208, two software distribution jobs and some device parameter settings were submitted in advance.

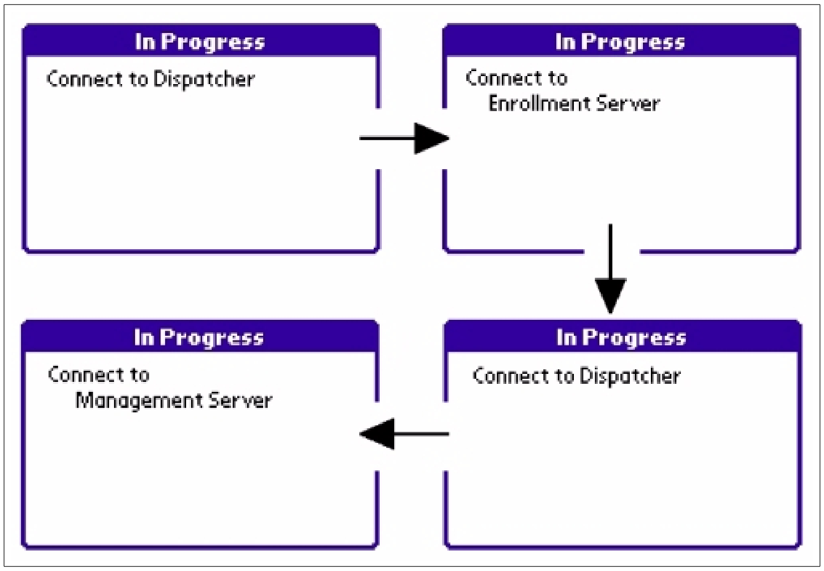
The following events need to take place:

1. The device is enrolled to the Device Manager database.
2. Software distribution jobs are submitted to this device.
3. The initial device configuration job is submitted. This causes the change in the network settings of the Palm device and device agent.

Following is a description of how the user connects to the Device Manager server, and how the jobs look to the user.

These tasks need to be completed to communicate with the Device Manager server:

1. Place the Palm device on the Palm modem, and connect the phone cable.
2. Click the **DevAgent** icon or **connect** button in the **PalmAgent** window of the device. The device agent program dials up the server.
3. After successfully connecting to the Device Manager server, the windows shown in Figure 106 are displayed.


In Progress

Connect to Dispatcher

In Progress

Connect to
Enrollment Server

In Progress

Connect to Dispatcher

In Progress

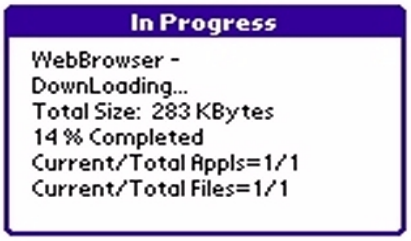
Connect to
Management Server

Figure 106. Agent displays at the first connection

In this case, the device is new to the system, so it is initially redirected to the enrollment server to be enrolled in the Device Manager database. It is then redirected to the device management server because there are more jobs submitted to this device.

4. The software distribution job for the Web browser is then submitted. This job is set so that users do not select the package or application, so the Web browser software is installed to the device with no user operation.

   Figure 107 on page 218 shows the progress of the software downloading.

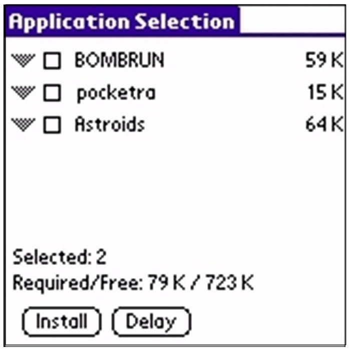


In Progress

WebBrowser -
DownLoading...
Total Size: 283 KBytes
14 % Completed
Current/Total Appls=1/1
Current/Total Files=1/1

Figure 107. Downloading the Web browser software

5. After downloading the Web browser software, another software distribution job starts. This job is set so that users can select the software to be installed. The application selection window, shown in Figure 108, is displayed.



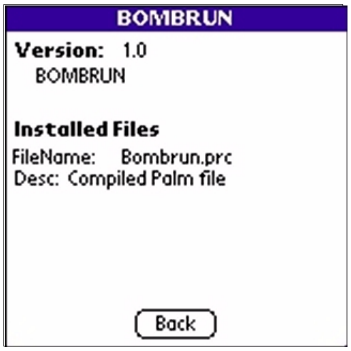
Application Selection

□	BOMBRUN	59 K
□	pocketra	15 K
□	Astroids	64 K

Selected: 2
Required/Free: 79 K / 723 K

Figure 108. Application selection window

6. Click the triangle to the left of the application name to display the description window of the application you selected, as shown in Figure 109.



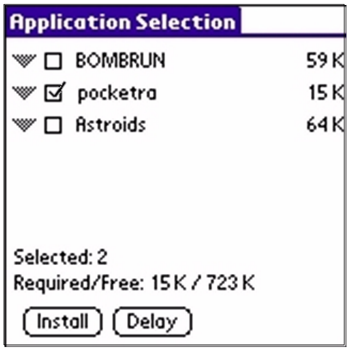
BOMBRUN

Version: 1.0
BOMBRUN

Installed Files
FileName: Bombrun.prc
Desc: Compiled Palm file

Figure 109. Description of the application

7. To install one of the software packages listed in the Application Selection window, shown in Figure 110, click the square to the left of the application name and click **Install**.



Application Selection

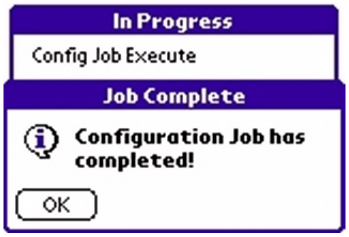
□	BOMBRUN	59 K
☑	pocketra	15 K
□	Astroids	64 K

Selected: 2
Required/Free: 15 K / 723 K

Figure 110. Software selection window

8. After downloading the selected software, the initial device configuration job is submitted to this device.

Figure 111 on page 220 shows that the device configuration job completed successfully.


In Progress

Config Job Execute

Job Complete

Configuration Job has completed!

Figure 111. Job completed

The device agent program disconnects from the network automatically after all the jobs have completed.

7.4 Device configuration

Each device class provides a configuration template file that describes the device-specific configuration parameters required by the devices it manages. This configuration template file defines the parameters, their syntax, and information to assist the Device Manager console in building a graphical user interface (GUI) to present the configuration parameters to an administrator. Device Manager console allows the configuration parameters to be viewed and modified for a specific device or for the device class.

7.4.1 Planning for device configuration

The system administrator uses the Device Manager console or an application to create a device configuration job and submit the job for processing.

At first, you need to know the flow to submit a device configuration job.

Figure 112 shows the steps you need to complete to modify device parameters for pervasive devices.

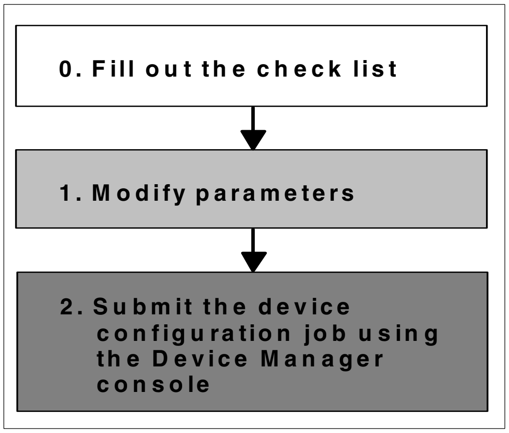

0. Fill out the check list
1. Modify parameters
2. Submit the device configuration job using the Device Manager console

Figure 112. Device configuration job flow

When planning for device configuration, we recommend that you first create and complete a checklist like the one shown in Table 33.

Table 33. Device configuration check list

Items	Description	Valid value
Target device class	Target device you want to modify its configuration	- Aero 8000 - Iad - Palm - Wince
Target device	Target device you want to modify its configuration	- All devices of a device class - A single device
Activation date	Date job become in service	YYYY MM/DD hh:mm
Expiration date	Date job become out of service	YYYY MM/DD hh:mm
Configuration Parameters	LabelKey to be modified	Depends on the parameter

Some configuration parameters are device class unique. Refer to the online manual *Tivoli Internet and Personalized Services Manager Device Manager: Device Plug-in Notes* to learn more about each device parameter.

7.4.2 Modifying the device configuration parameters

A service provider can set the device configuration parameters for each device, or for all devices of a device class, at any time. Normally, the configuration parameters are set as an initial job by the service provider.

7.4.2.1 The template file

When you prepare your customized configuration template file or modify an existing one, refer to the following resources:

- Lists of configuration parameters in *Tivoli Internet and Personalized Services Manager Device Manager: Device Plug-in Notes*
- Configuration template file for each device class in the `/usr/lpp/TivDMS/bin` directory on the Device Manager server
- Configuration template file format information in the online manual *Tivoli Personalized Services Manager Device Manager: Developer's Guide*

The format of the configuration template file is common for all device classes. As described in Chapter 6, "Application programming interface" on page 171, Device Manager provides a command line utility, the `devclasscfg.sh` command script, to install the configuration template files when the device class plug-in is configured. For more information about the `devclasscfg.sh` command script, refer to 4.2.2, "Managing a device class" on page 112.

Note

In Device Manager 1.1, device configuration parameters may be set for individual devices or all devices of a device class; they cannot be set by realm. Until this is corrected, the desired realm-specific configuration parameters may programmatically be set to the desired configuration for the individual devices in the realm. A realm-specific device configuration job can then be submitted.

7.4.2.2 Device Manager database tables

When you modify an existing device configuration using Device Manager console or device management API, only modified parameters are stored in the Device Manager database. It means there is no entry in the DEVICE_PARM table and the DEVICE_CLASS_PARM table.

- DEVICE_PARM table is a table that includes information about the modifications you made for each device.
- DEVICE_CLASS_PARM table is a table that includes information about the modifications you made for a device class.

For detailed information about the previously mentioned tables, refer to Chapter 5, "Device Manager database" on page 145.

You can make changes to device class parameters through the Device Manager console as described in the next section.

7.4.2.3 Modifying parameters using Device Manager console

Figure 113 shows an empty DEVICE_CLASS_PARM table.

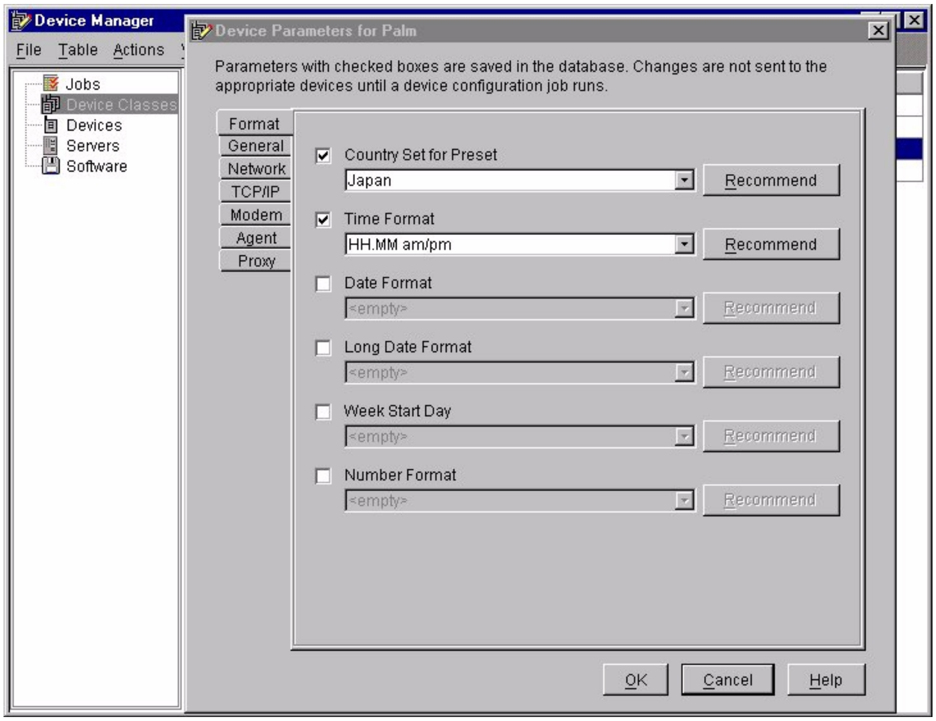
```
SQL> select * from DEVICE_CLASS_PARM;

no rows selected

SQL>
```

Figure 113. Example of empty DEVICE_CLASS_PARM table

To modify parameters using Device Manager console, complete the following steps:

1. In the right pane of the console, highlight the device class you want to modify (in our case we selected Palm device class). Next, right-click it and select **Device Parameters** from the context menu. The Device Parameters For Palm window is displayed.
2. Select and modify the desired attributes.
3. To save this information and close the window, click **OK**.

Figure 114 on page 224 shows the Device Parameters For Palm window previously described.



Parameters with checked boxes are saved in the database. Changes are not sent to the appropriate devices until a device configuration job runs.

☑ Country Set for Preset
Japan

☑ Time Format
HH.MM am/pm

□ Date Format
<empty>

□ Long Date Format
<empty>

□ Week Start Day
<empty>

□ Number Format
<empty>

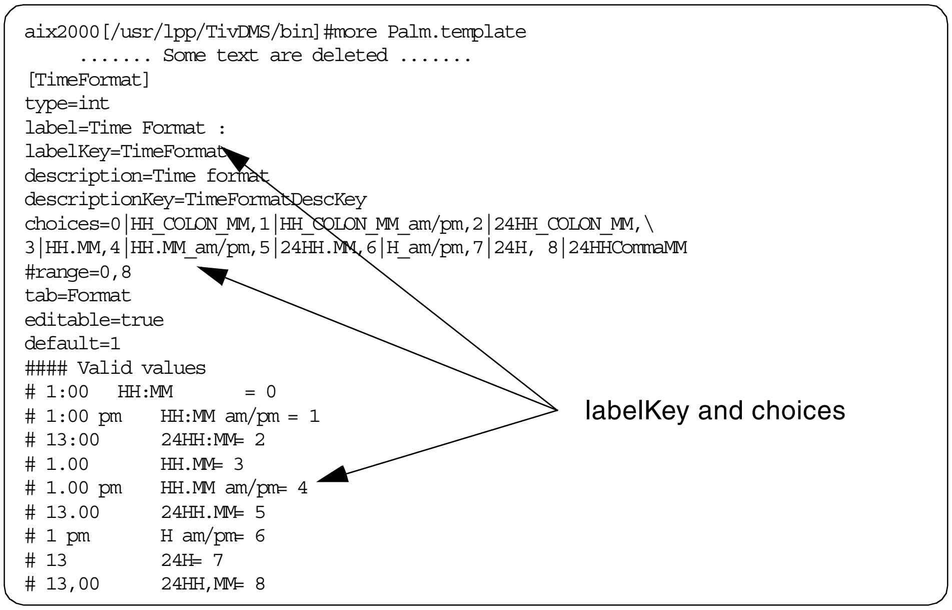
Figure 114. Modifying the device class parameters

Now, you can retrieve modified parameters from the DEVICE_CLASS_PARM table in the Device Manager database as shown in Figure 115.

```
SQL> select * from DEVICE_CLASS_PARM;

DEVICE_CLASS_ID        PARM_KEY       PARM_VALUE    LAST_MODI
--------------------------------------------------------------------------------
       1000000         TimeFormat              4     03-NOV-00


DEVICE_CLASS_ID        PARM_KEY       PARM_VALUE    LAST_MODI
--------------------------------------------------------------------------------
       1000000        Preset                  13    03-NOV-00

SQL>
```

Figure 115. Example of the DEVICE_CLASS_PARM table with entries

The meanings of the entries in the PARM_VALUE field are described in the template files, as shown in Figure 116. The PARM_KEY and the PARM_VALUE depend on the device classes.



```
aix2000[/usr/lpp/TivDMS/bin]#more Palm.template
     ....... Some text are deleted .......
[TimeFormat]
type=int
label=Time Format :
labelKey=TimeFormat
description=Time format
descriptionKey=TimeFormatDescKey
choices=0|HH_COLON_MM,1|HH_COLON_MM_am/pm,2|24HH_COLON_MM,\
3|HH.MM,4|HH.MM_am/pm,5|24HH.MM,6|H_am/pm,7|24H, 8|24HHCommaMM
#range=0,8
tab=Format
editable=true
default=1
#### Valid values
# 1:00   HH:MM       = 0
# 1:00 pm    HH:MM am/pm = 1
# 13:00      24HH:MM= 2
# 1.00       HH.MM= 3
# 1.00 pm    HH.MM am/pm= 4
# 13.00      24HH.MM= 5
# 1 pm       H am/pm= 6
# 13         24H= 7
# 13,00      24HH,MM= 8
```

Figure 116. Palm template file

After modifying the parameters, you need to submit a job for device configuration. Make sure that Device Configuration is selected in the Job type field.

7.5 Sample scenario for device configuration

This section explains how to modify the device parameters and submit device configuration jobs for the Palm device.

7.5.1 Scenario 2

An ISP is moving from the pilot to the production stage. In the pilot stage, a single Device Manager server was used. In the production environment, scalability and fault tolerance are more of a concern, so more DM servers and a Network Dispatcher is added. The DM server address now must change from the address of the pilot DM server to that of the Network Dispatcher cluster for the Device Manager machines.

This section shows you how to create device configuration jobs according to Figure 112 on page 221.

7.5.2 System environment

The system environment is almost the same as the one described in Figure 98 on page 209. The only difference is that now we have a new Device Manager server and a Network Dispatcher machine. Figure 117 shows the image of this device configuration job.

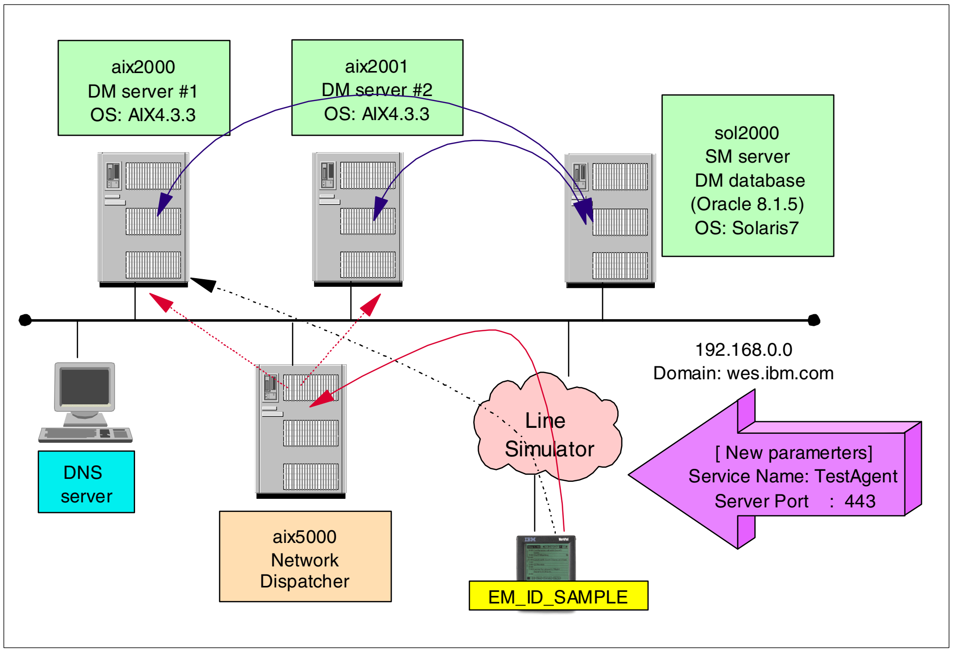


Figure 117. Device configuration for a Palm device

7.5.3 Server operations

To change the Device Manager server and connection port number for a Palm device named EM_ID_SAMPLE, we had to change the following four parameters:

- **DMSAddress**: Device Manager server address. This device was connected to server aix2000.wes.ibm.com. After the device configuration job, aix5000.wes.ibm.com is this device’s new Device Manager server. Actually, this is the address of the Network Dispatcher cluster for the Device Manager machines.
- **SSLOn**: SSL Enable.
- **DMSPort**: Device Manager server port number (integer value). The default port is 80, and the new port is 443.

- **ServiceName**: Service (Network interface) name to be used. The default service name is DevAgent, and the new service name is TestAgent.

Network Dispatcher

WebSphere Edge Server Load Balancer (Network Dispatcher) provides dynamic load balancing, scalability, and high availability for servers, boosting overall server performance by automatically finding the optimal server within a group of servers to handle each incoming request. It can be used with Web servers, e-mail servers, distributed parallel database queries, and other Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) applications.

For more information about Network Dispatcher, see the Web site:
`http://www.ibm.com/software/network/dispatcher`

7.5.3.1 Filling in the check list

This section describes the system administration device configuration steps.

Based on the sample scenario mentioned in 7.5.1, “Scenario 2” on page 225, we completed the checklist shown in Table 34.

Table 34. Sample scenario device configuration checklist

Item	Valid value
Target device class	Palm
Target device	a device EM_ID_SAMPLE
Activation date	2000 11/02 12:00
Expiration date	2000 12/31 12:00
Parameter(s)	DMSAddress: aix5000.wes.ibm.com
	SSLOn: 1
	DMSPort: 443
	ServiceNAme: TestAgent

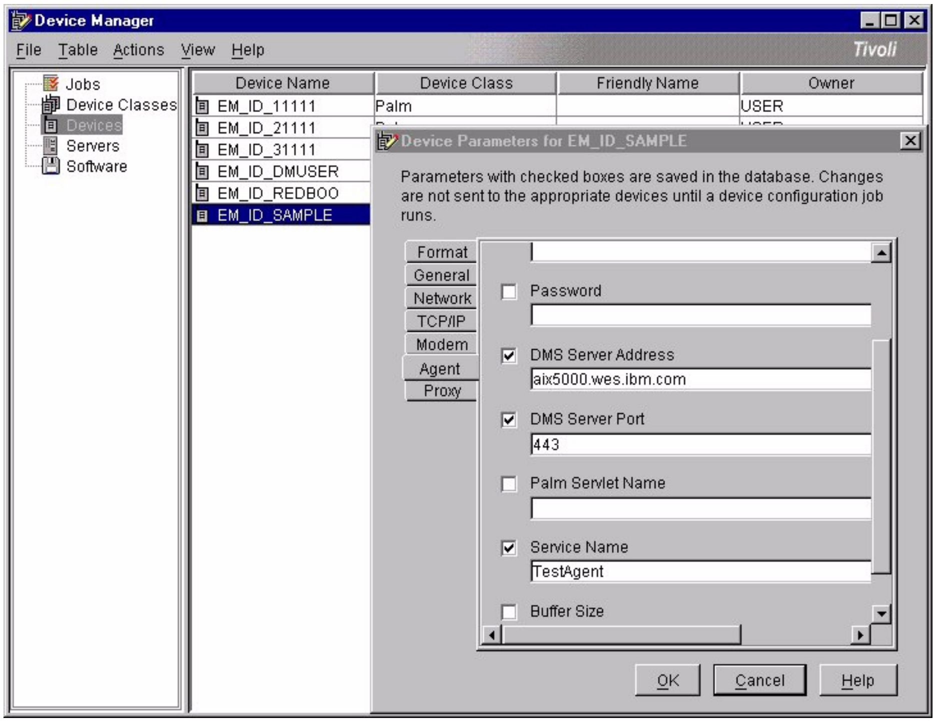
7.5.3.2 Modifying the parameters and submit job

1. Make sure a device, named EM_ID_SAMPLE, is highlighted in the right pane of the console. Then, right-click it and select **Device Parameters**

from the context menu. The Device Parameters For EM_ID_SAMPLE window is displayed.

2. Select the **Agent** tag, and then select the boxes to the left of the parameters to be modified. Next, change the parameters.

Figure 118 shows the Device Parameter window when the Agent tag is selected.



Device Name	Device Class	Friendly Name	Owner
EM_ID_11111	Palm		USER
EM_ID_21111			
EM_ID_31111			
EM_ID_DMUSER			
EM_ID_REDBOO			
EM_ID_SAMPLE			

Parameters with checked boxes are saved in the database. Changes are not sent to the appropriate devices until a device configuration job runs.

☐ Password

☑ DMS Server Address: aix5000.wes.ibm.com

☑ DMS Server Port: 443

☐ Palm Servlet Name

☑ Service Name: TestAgent

☐ Buffer Size

Figure 118. Device Parameter window

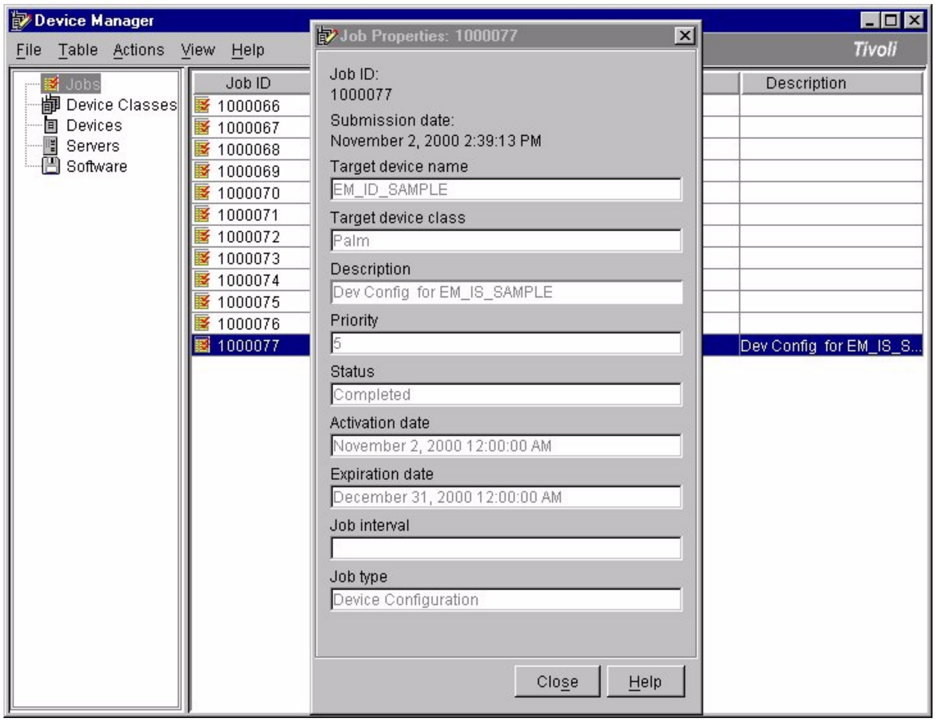
3. To save this information and close the window, click **OK**.

4. To enable the changes, you need to submit a device configuration job to the target device. Make sure a device, named EM_ID_SAMPLE, is highlighted in the right pane of the console. Then right-click it and select **Submit Job** from the context menu. The New Job Properties window is displayed.

5. Complete the necessary information like Activation date field, and make sure Job Type is Device Configuration. Click **OK**.

When a target device connects to the Device Manager server, the device configuration job runs and the modified parameters are sent to the appropriate device. Then the status of the job changes to Completed.

Figure 119 shows the job status after the device configuration job executed.



Job ID	Description
1000066	
1000067	
1000068	
1000069	
1000070	
1000071	
1000072	
1000073	
1000074	
1000075	
1000076	
1000077	Dev Config for EM_IS_S...

Job ID:
1000077

Submission date:
November 2, 2000 2:39:13 PM

Target device name: EM_ID_SAMPLE

Target device class: Palm

Description: Dev Config for EM_IS_SAMPLE

Priority: 5

Status: Completed

Activation date: November 2, 2000 12:00:00 AM

Expiration date: December 31, 2000 12:00:00 AM

Job interval:

Job type: Device Configuration

Figure 119. Device configuration job property

7.5.4 Client operations

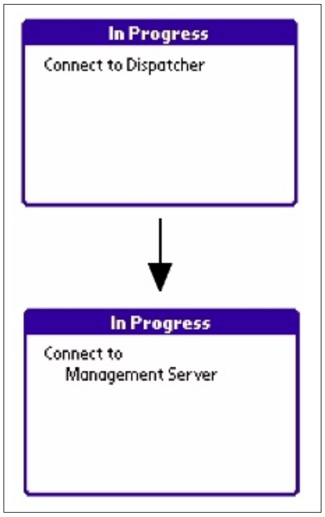
According to the scenario mentioned in 7.5.1, “Scenario 2” on page 225, some device parameter settings to communicate with the Device Manager server through a Network Dispatcher using SSL, are submitted in advance.

The following information describes the user operations for connecting to the Device Manager server and how the job looks to the user.

The tasks to communicate with the Device Manager server are listed here:

1. Click the **DevAgent** icon or **Connect** button in the PalmAgent window to invoke the device agent program.

2. The device is already enrolled to the Device Manager database at this time, so the device is automatically redirected to the Device management server as shown in Figure 120.



In Progress

Connect to Dispatcher

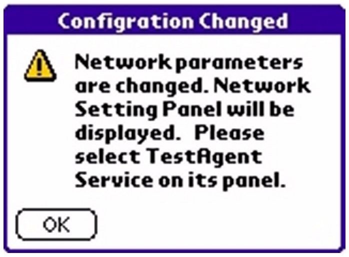
In Progress

Connect to
Management Server

Figure 120. The device is directly redirected to the management server

3. The device configuration job is performed. This job changes some network settings on the device.

4. The window shown in Figure 121, indicating that the network parameters have been changed, is displayed.

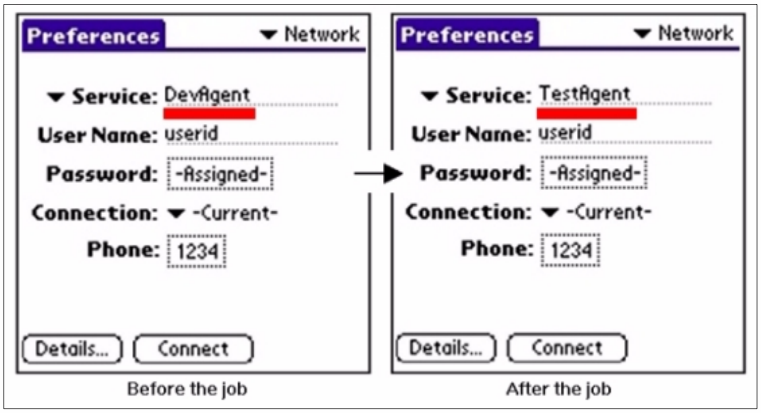


Configration Changed

Network parameters are changed. Network Setting Panel will be displayed. Please select TestAgent Service on its panel.

Figure 121. The message after the device configuration job

5. Now that the network service that the device agent program uses is changed to “TestAgent”, select **TestAgent** in the network preference window shown in Figure 122.



Before the job

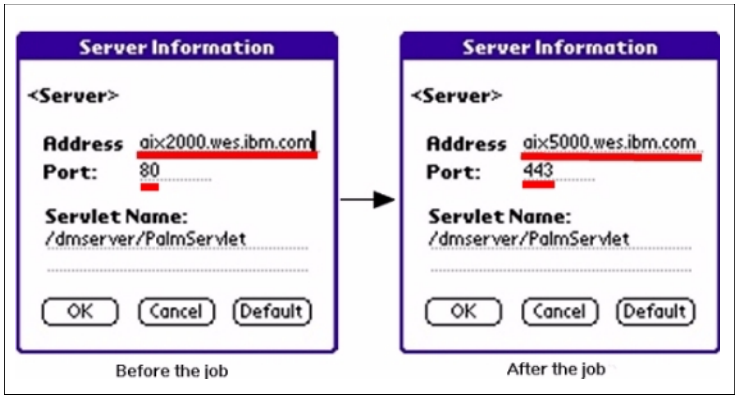
Service: DevAgent
User Name: userid
Password: -Assigned-
Connection: -Current-
Phone: 1234

After the job

Service: TestAgent
User Name: userid
Password: -Assigned-
Connection: -Current-
Phone: 1234

Figure 122. The service name has changed

6. The device configuration job also changed the Device Manager server address and SSL setting. This can be confirmed from the Device Manager main window. Click the **List** menu icon and select **Options** -> **Server Settings**. Then you can see the window shown in Figure 123, with the parameters already changed.



Before the job

<Server>
Address aix2000.wes.ibm.com
Port: 80
Servlet Name:
/dmserver/PalmServlet

After the job

<Server>
Address aix5000.wes.ibm.com
Port: 443
Servlet Name:
/dmserver/PalmServlet

Figure 123. The Device Manager server address has changed

Now the device agent settings are changed to use SSL. The next time you connect to the Device Manager server, the communication will be established with SSL based on the settings.

Appendix A. Device parameters

This appendix provides the list of all the device parameters for the built-in device plug-ins. These are defined in each plug-in specific device parameter file on the Device Manager server, whose format is described in Chapter 6, “Application programming interface” on page 171. You can set values to these parameters from the Device Manager console or device management API. For detailed steps to use the DM console, refer to Chapter 7, “Using the DM functions” on page 193. For information about device management API, refer to Chapter 6, “Application programming interface” on page 171.

A.1 Parameters for the Palm plug-in

Parameters for the Palm devices are classified into seven tabs:

- Format
- General
- Network
- TCP/IP
- Modem
- Agent
- Proxy

The first five tabs have parameters for Palm OS preferences, and the latter two tabs have parameters for Palm device agent settings.

A.1.1 Format tab

The parameters that can be set in the Format tab (Table 35 on page 234) correspond to those in **Prefs -> Format** on the Palm device.

Table 35. Palm plug-in: Parameters in the Format tab

Label name	**Palm keyword**	**Valid values**	
Country Set for Preset	Preset	Integer Changing the configuration value for Preset has no effect on the values for the other Format keywords. **23** (UnitedStates) is the default. The valid values are:	
		0: Australia **1:** Austria **2:** Belgium **3:** Brazil **4:** Canada **5:** Denmark **6:** Finland **7:** France **8:** Germany **9:** HongKong **10:** Iceland **11:** Ireland	**12:** Italy **13:** Japan **14:** Luxembourg **15:** Mexico **16:** Netherlands **17:** NewZealand **18:** Norway **19:** Spain **20:** Sweden **21:** Switzerland **22:** UnitedKingdom **23:** UnitedStates
Time Format	TimeFormat	Integer **1** (HH:MM am/pm) is the default. Valid values are as follows. The examples of the format are in parentheses.	
		0: HH:MM (1:00) **1:** HH:MM am/pm (1:00 pm) **2:** 24HH:MM (13:00) **3:** HH.MM (1.00) **4:** HH.MM am/pm (1.00 pm) **5:** 24HH.MM (13.00) **6:** H am/pm (1 pm) **7:** 24H (13) **8:** 24HH,MM (13,00)	

Label name	Palm keyword	Valid values
Date Format	DateFormat	Integer **0** (M/D/Y) is the default. Valid values are as follows. The examples of the format are in parentheses.
		0: M/D/Y (12/31/95) **1:** D/M/Y (31/12/95) **2:** D.M.Y (31.12.95) **3:** D-M-Y (31-12-95) **4:** Y/M/D (95/12/31) **5:** Y.M.D (95.12.31) **6:** Y-M-D (95-12-31) **7:** MDY Long w/ comma (Dec 31, 1995) **8:** DMY Long (31 Dec 1995) **9:** DMY Long w/ dot (31. Dec 1995) **10:** DMY Long No Day (Dec 1995) **11:** DMY Long w/ comma (31 Dec, 1995) **12:** YMD Long w/ Dot (1995.12.31) **13:** YMD Long w/ Space (1995 Dec 31) **14:** MY Med (Dec '95) **15:** MY No Post (Dec 95)
Long Date Format	LongDateFormat	Integer **7** (MDY Long w/ comma) is the default. Valid values are as follows. The examples of the format are in parentheses.
		7: MDY Long w/ comma (Dec 31, 1995) **8:** DMY Long (31 Dec 1995) **9:** DMY Long w/ dot (31. Dec 1995) **10:** DMY Long No Day (Dec 1995) **11:** DMY Long w/ comma (31 Dec, 1995) **12:** YMD Long w/ Dot (1995.12.31) **13:** YMD Long w/ Space (1995 Dec 31)
Week Start Day	WeekStartDay	Integer **0** (Sunday) is the defalut value. Valid values are as follows.
		0: Sunday **1:** Monday **2:** Tuesday **3:** Wednesday **4:** Thursday **5:** Friday **6:** Saturday

Label name	Palm keyword	Valid values
Number Format	NumberFormat	Integer **0** (CommaPeriod) is the default value. Valid values are as follows. The numbers in parentheses indicate examples of the format.
		0: CommaPeriod (1,000.00) **1:** PeriodComma (1.000,00) **2:** SpaceComma (1 000,00) **3:** ApostrophePeriod (1’000.00) **4:** ApostropheComma (1’000.00)

A.1.2 General tab

The parameters that can be set in the General tab (Table 36) correspond to those in **Prefs -> General** on the Palm device.

Table 36. Palm plug-in: Parameters in the General tab

Label name	Palm keyword	Valid values
Set Date/Time	SetDateTime	Use the keyword CURRENT or a string value. Default value is **CURRENT**.
Auto-Off Timer (min.)	AutoOffTimer	Auto power off timer in minutes (integer) **2** (min.) is the default value. Set to 0 to disable the timer.
System Sound Setting	SystemSound	Integer **High** is the default setting. There are four choices for this keyword. The values actually set to Palm device are in parentheses.
		No sound (0) **Low** (8) **Medium** (16) **High** (64)
Alarm Sound Setting	AlarmSound	Integer **High** is the default setting. There are four choices for this keyword (the same as SystemSound).
Game Sound Setting	GameSound	Integer **High** is the default setting. There are four choices for this keyword (the same as SystemSound).

A.1.3 Network tab

The parameters that can be set in the Network tab (Table 37) correspond to User name and Password keys in **Prefs -> Network** on the Palm device.

Table 37. Palm plug-in: Parameters in the Network tab

Label name	Palm keyword	Valid values
User Name for PPP	UserName	These keywords should be String and are used for a PPP connection. They are separate from the PalmUserID and PalmPassword in the Agent tab, which are used for the Subscription Manager component single sign-on.
Password	Password	

A.1.4 TCP/IP tab

The parameters that can be set in the TCP/IP tab (Table 38) correspond to those in **Prefs -> Format -> Details ...** on the Palm device.

Table 38. Palm plug-in: Parameters in the TCP/IP tab

Label name	Palm keyword	Valid values
PPP queries for DNS address	DNSQuery	Integer **On** is the default value. Valid values are as follows:
		On (1) **Off** (0)
Primary DNS	PrimaryDNS	String
Secondary DNS	SecondaryDNS	String

A.1.5 Modem tab

The parameters that can be set in the Modem tab (Table 39) correspond to the phone key in **Prefs -> Network** on the Palm device.

Table 39. Palm plug-in: Parameters in the Modem tab

Label name	Palm keyword	Valid values
Modem Phone Number	ModemPhone	String

A.1.6 Agent tab

The parameters that can be set in the Agent tab (Table 40) are for the Palm device agent. DMSAddress, DMSPort, and PalmServletName have a GUI for you to set them manually.

Table 40. Palm plug-in: Parameters in the Agent tab

Label name	Palm Keyword	Valid Values
SSL On	SSLOn	**Off** is the default setting. Valid values are as follows:
		On (1) **Off** (0)
User ID	PalmUserID	String. The subscriber's user ID as defined in the Subscription Manager component
Password	PalmPassword	String. The subscriber's password as defined in the Subscription Manager component.
DMS Server Address	DMSAddress	Either hostname or IP address
DMS Server Port	DMSPort	Integer. Default is **80** (in case non-SSL) or **443** (in case SSL). Palm agent will use SSL connection if SSLOn is On and DMSPort is 443.
Palm Servlet Name	PalmServletName	String Default is **/dmserver/PalmServlet**.
Service Name	ServiceName	String Network interface name to use for Palm OS agent program. Default is **DevAgent**.
Buffer size	BufferSize	Integer Agent receive buffer size in KB. Default is **8** (KB). Valid range is 4 through 24 with SSL off, 4 through 8 with SSL on.

A.1.7 Proxy tab

The parameters that can be set in the Proxy tab (Table 41) are for the Palm device agent. The device agent has a GUI for you to set them manually.

Table 41. Palm plug-in: Parameters in the Proxy tab

Label name	Palm keyword	Valid values
ProxyEnable	ProxyEnable	Integer **Off** is the default. Valid values are as follows:
		On (1) **Off** (0)
ProxyAddress	ProxyAddress	String Specify the hostname of the proxy server.
ProxyPort	ProxyPort	Integer Default is **80**.

A.2 Parameters for the Aero 8000 plug-in

The parameters for the Aero 8000 devices are classified into six tabs:

- PPP
- TCP/IP
- Browser
- Agent Setting
- Mailer
- Mgmt

Each parameter for Aero 8000 has an Aero 8000 keyword name, which may correspond to a common key. The Aero 8000 agent program recognizes the configuration parameters by the Aero 8000 keyword name. The common key name is defined by Tivoli Personalized Services Manager and does not depend on the Aero 8000 device. Common key names are converted to Aero 8000 keyword names by the Aero 8000 device configuration job class. Therefore, we recommend that you use the Aero 8000 keyword names, not the common key names. For this reason, this appendix does not introduce the common keys. For more detailed information on the common keys, refer to *Tivoli Personalized Services Manager Device Manager: Developer's Guide* and *Tivoli Internet and Personalized Services Manager Device Manager: Device Plug-in Notes* included in Device Manager online documents.

The mapping of Aero 8000 keywords and common keys is not one-to-one. When using keyword names, use either all Aero 8000 keyword names or all common key names. Do not use a mixture of names from the two keyword sets.

Device unique configuration parameters

There are some configuration parameters that are unique for each Aero 8000 device, such as the user ID, password, and mail server information. The device unique configuration parameters are commented out in the aero8000.template file and do not appear on the Device Manager console.

If an administrator wants to set values for the device unique configuration parameters, edit the configuration parameter file so these fields appear on the Device Manager user interface.

For the Aero 8000, the device unique configuration parameters are:

- PPP tab
  - ppp.user
  - ppp.pass
- Mailer tab
  - pop3.uid
  - mail.address

Descriptions of these device unique configuration parameters appear in following parameter descriptions.

A.2.1 PPP tab

Table 42 lists the parameters in the PPP tab. The keywords ppp.user and ppp.pass are disabled in the template file by default. If you want to use them, remove the comment symbols from them in the aero8000.template file placed in the `/usr/lpp/TivDMS/bin` (AIX) or `/opt/TivDMS/bin` (Solaris) directory before you register the Aero 8000 device class to the Device Manager database.

Table 42. Aero 8000 plug-in: Parameters in the PPP tab

Label name	Aero8000 keyword	Valid values
PPP access point	ppp.dial	String. The telephone number of PPP access point.
PPP user ID	ppp.user	String

Label name	Aero8000 keyword	Valid values
PPP password	ppp.pass	String

A.2.2 TCP/IP tab

Table 43 lists the parameters in the TCP/IP tab.

Table 43. Aero 8000 plug-in: Parameters in the TCP/IP tab

Label name	Aero8000 keyword	Valid values
DNS primary	net.dns1	String
DNS secondary	net.dns2	String

A.2.3 Browser tab

Table 44 lists the parameters in the TCP/IP tab.

Table 44. Aero 8000 plug-in: Parameters in the Browser tab

Label name	Aero8000 keyword	Valid values
Start page	bsr.startpage	String URL of Start page
HTTP proxy server	bsr.proxyaddr	String Proxy server IP address
HTTP proxy port	bsr.proxyport	Integer Proxy server port number **80** is the default.
PCT setting	pct.enable	String PCT setting for browser Valid values for this parameter are as follows:
		ON OFF
SSL2 setting	ssl2.enable	String SSL2 setting for browser Valid values for this parameter are as follows:
		ON OFF

Label name	Aero8000 keyword	Valid values
SSL3 setting	ssl3.enable	String Valid values for this parameter are as follows:
		ON OFF

A.2.4 Agent tab

Table 45 lists the parameters in the Agent tab.

Table 45. Aero 8000 plug-in: Parameters in the Agent tab

Label name	Aero8000 keyword	Valid values
SSL setting	ssl.enable	String. Valid values for this parameter are as follows:
		ON OFF

A.2.5 Mailer tab

Table 46 lists the parameters in the Mailer tab. The keywords pop3.uid and mail.address are disabled in the template file by default. If you want to use them, remove the comment symbols from them in the aero8000.template file placed in the `/usr/lpp/TivDMS/bin` (AIX) or `/opt/TivDMS/bin` (Solaris) directory before you register the Aero 8000 device class to the Device Manager database.

Table 46. Aero 8000 plug-in: Parameters in Mailer

Label name	Aero8000 keyword	Valid values
POP3 server	pop3.server	String For example, pop3svr.tivoli.com
SMTP server	smtp.server	String For example, smtpsvr.tivoli.com
POP3 user ID	pop3.uid	String
Mail account	mail.address	String Mail account for return e-mail address Device unique.

A.2.6 Mgmt tab

Table 47 lists the parameters in the Agent tab. As described in the table, please note that some keys are not applied to Pocket Internet Explorer. Refer to the last row of Table 47.

Table 47. Aero 8000 plug-in: Parameters in the Mgmt tab

Label name	Aero8000 keyword	Valid values
Mgmt server	mgmt.server	String Management server URL For example, http://9.3.4.1/mgsrvlt
Polling timer	mgmt.pollingtimer	Integer Polling timer in hours **10** is the default. Valid range: 1 to 65535
Agent run mode	mgmt.agentrunmode	String Determines if the agent program runs after all jobs are complete. Valid values are as follows.
		ON = Leaves the agent program running. OFF = Closes the agent program after the jobs are run.
Proxy enable	mgmt.proxy.enable	**OFF** is the default. Valid values are as follows.
		ON OFF
Proxy server address	mgmt.proxy.addr	For example, www.proxy.com
Proxy server port	mgmt.proxy.port	**80** is the default.
TPSM user ID	mgmt.auth.user	String The ID defined in the Subscription Manager component.
TPSM password	mgmt.auth.pass	String. The password in the Subscription Manager component.

Label name	Aero8000 keyword	Valid values
Note: The settings for mgmt.proxy.addr and mgmt.proxy.port do not apply to the Pocket Internet Explorer browser or any other Web browser. The proxy settings of the Pocket Internet Explorer browser are independent from these settings. The subscriber's password, as defined in the Subscription Manager component, is entered by the user when the device agent program starts. The password is stored in memory only, so that no one can peek at the password in the registry.		

A.3 Parameters for the Windows CE plug-in

The parameters for the Windows CE devices are classified into six tabs:

- PPP
- TCP/IP
- Browser
- Agent Setting
- Mailer
- Mgmt

For detailed information on Windows CE device parameters, refer to A.2, “Parameters for the Aero 8000 plug-in” on page 239, because the set of parameters and their attributes are the same as the Aero 8000 plug-in’s.

A.4 Parameters for the NetVista Internet appliance plug-in

The parameters for the Windows CE devices are classified into 15 tabs:

- Log Service
- HTTPD Service
- Device Agent
- Browser Default
- Time Server
- Printing
- Network Printer 1
- Network Printer 2
- Network Printer 3
- Dialup
- Ethernet
- Syslog
- Email

- Connection Manager
- PAM Drive

We call NetVista Internet Appliance “IAD” in this appendix.

A.4.1 Log Service tab

Table 48 lists the parameters in Log Service tab.

Table 48. IAD plug-in: Parameters in the Log Service tab

Description	**Internet Appliance keyword**	**Valid values**
Log Entry Size	/s/smf/LogService/ LogSize	Integer. Maximum number of recordable logs. **100** is the default.
Log Threshold	/s/smf/LogService/ LogThreshold	Integer. This sets the log level. **2** is the default. Valid range is from 1 to 4.

A.4.2 HTTPD Service tab

Table 49 lists the parameters in the HTTPD Service tab.

Table 49. IAD plug-in: Parameters in the HTTPD Service tab

Label name	**Internet Appliance keyword**	**Valid values**
HTTP Port	/s/smf/HttpService /HttpPort	Integer **80** is the default.
HTTP Maximum Threads	/s/smf/HttpServce/ MaxThreads	Integer **6** is the default.
HTTP Thread Priority	/s/smf/HttpService /TreadPriority	Integer Running priority of threads in HTTPD **4** is the default.

A.4.3 Device Agent tab

Table 50 lists the parameters in the Device Agent tab.

Table 50. IAD plug-in: Parameters in the Device Agent tab

Label name	Internet Appliance keyword	Valid values
Management Server URL	/PvCAgent/ServerAddress	For example, https://9.3.8.100/mgmtservlet
Enrollment Server URL	/PvCAgent/EnrollServerAddress	For example, http://9.3.8.200/enrollment
Proxy Address	/PvCAgent/ProxyAddress	For example, proxy.company.com
Proxy Port	/PvCAgent/ProxyPort	Integer For example, 8080
Proxy Enablement	/PvCAgent/ProxyEnable	**False** is the default. Valid values are as follows.
		True False
Reboot Window-Start	/PvCAgent/RebootStart	String. The default is RebootStart = RebootEnd, so a reboot occurs. For example, 2:00
Reboot Window-End	/PvCAgent/RebootEnd	String The default is RebootStart = RebootEnd, so a reboot occurs. For example, 4:59
Polling Interval	/PvCAgent/PollingInterval	Integer Job polling interval in hours **4** is the default. 1 hour is the minimum and valid range is 1 to 65535.
Polling Enablement	/PvCAgent/PollingEnable	**True** is the default setting. Valid values are as follows.
		True False

Label name	Internet Appliance keyword	Valid values
Polling Window-Start	/PvCAgent/PollingStart	String The default is PollingStart = PollingEnd, so polling occurs. For example, 2:00
Polling Window-End	/PvCAgent/PollingEnd	String The default is PollingStart = PollingEnd, so polling occurs. For example, 3:59
Basic Authentication User ID	/PvCAgent/AuthenticationUserID	String
Basic Authentication Password	/PvCAgent/AuthenticationPassword	String
UI panel timeout	/PvCAgent/PanelTimeout	Integer Time-out value to close user interface **20** (seconds) is the default.

A.4.4 Browser Default tab

Table 51 shows the parameters in the Device Agent tab.

Table 51. IAD plug-in: Parameters in the Browser Default tab

Label name	Internet Appliance keyword	Valid values
Cache Size	/shell/defaultSetting/WWW Section/Cache Size	Integer. **5000** is the default (KB).
Image Cache Size	/shell/defaultSetting/WWW Section/Image Cache Size KB	Integer **1024** is the default (KB).
Page Cache Size	/shell/defaultSetting/WWW Section/Page Cache Size	Integer **4** is the default.
Max Connections	/shell/defaultSetting/WWW Section/Max Connections	Integer Maximum number of connections **5** is the default.

Label name	**Internet Appliance keyword**	**Valid values**
Use Proxy Server	/shell/defaultSetting/WWW Section/Use Proxy Server	**True** is the default. Valid values are as follows.
		True False
Proxy Overrides	/shell/defaultSetting/WWW Section/Proxy Overrides	List of host names or IP addresses which bypass the currently selected proxy server. Use a comma to separate the list members. For example, www.internal.ibm.com
Socks Proxy Address	/shell/defaultSetting/WWW Section/Socks Proxy Address	For example, socks.ibm.com
Socks Proxy Port	/shell/defaultSetting/WWW Section/Socks Proxy Port	Integer **1080** is the default.
FTP Proxy Address	/shell/defaultSetting/WWW Section/FTP Proxy Address	For example, proxy.ibm.com
FTP Proxy Port	/shell/defaultSetting/WWW Section/FTP Proxy Port	Integer. **80** is the default.
HTTPS proxy address	/shell/defaultSetting/WWW Section/HTTPS Proxy Address	For example, proxy.ibm.com
HTTPS Proxy Port	/shell/defaultSetting/WWW Section/HTTPS Proxy Port	Integer **80** is the default.
HTTP Proxy Address	/shell/defaultSetting/WWW Section/HTTP Proxy Address	For example, proxy.ibm.com
HTTP Proxy Port	/shell/defaultSetting/WWW Section/HTTP Proxy Port	Integer **80** is the default.
Home Page	/shell/defaultSetting/WWW Section/Home Page	URL for home page. For example, www.ibm.com

A.4.5 Time Server tab

Table 52 lists the parameters in the Time Server tab.

Table 52. IAD plug-in: Parameters in the Time Server tab

Label name	Internet Appliance keyword	Valid values
Time Zone	/native/TimeZone	String Local time zone For example, jst-09
Time Server	/native/TimeServer	Time server address to adjust the system clock. For example, time.company.com

A.4.6 Printing tab

Table 53 shows the parameters in the Printing tab.

Table 53. IAD plug-in: Parameters in the Printing tab

Label name	Internet Appliance keyword	Valid values
Spool directory for Spooler	/native/Printer/Spooler/Directory	The directory used to create print spooler files. For example, /ram/spooler
Error Logging File Name	/native/Printer/ErrorLog	Specify log file for errors. For example, /logs/sys/errorlog

A.4.7 Network Printer 1 tab

Table 54 shows the parameters in the Network Printer 1 tab. Network Printer 2 tab and Network Printer 3 tab have the same set of parameters as Network Printer 1 tab. However each tab has its own keyword. For example, /native/NetworkPrinter1/RemoteMachineName is the keyword for Network printer machine name of Network Printer 1, and /native/NetworkPrinter2/RemoteMachineName is that of Network Printer 2. In this table, we describe only parameters in Network Printer 1 tab.

Table 54. IAD plug-in: Parameters in the Network Printer 1 tab

Label name	Internet Appliance keyword	Valid values
Remote Machine Name	/native/NetworkPrinter1/RemoteMachineName	Network printer machine name. For example, netprinter.company.com

Label name	Internet Appliance keyword	Valid values
Remote Machine Argument	/native/NetworkPrinter1/RemoteMachineArg	Arguments for network printer. For example, josa-ps
Data Stream	/native/NetworkPrinter1/DataStream	Valid values are as follows.
		ps pc1
Printer Name	/native/NetworkPrinter1/Name	Network printer short name. For example, InfoPrint21
Printer Description	/native/NetworkPrinter1/Description	Description of network printer. For example, Info Printer 21 in test lab
Spooler Device Name	/native/NetworkPrinter1/Spooler/DeviceName	Device name to send the spool file to. For example, /dev/null
Printer Queue Size	/native/NetworkPrinter1/QueueSize	Integer Maximum queue size in KB to be used for this printer. **5000** is the default (KB).
Spooler Directory	/native/NetworkPrinter1/SpoolDirectory	Directory where temporary files are created for spooler. For example, ram/spooler

A.4.8 Dialup tab

Table 55 lists the parameters in the Dialup tab.

Table 55. IAD plug-in: Parameters in the Dialup tab

Label name	Internet Appliance keyword	Valid values
Phone Number	/native/DialupPhoneNumber	For example, 555-1212
Userid	/native/DialupUserid	String Dialup user ID
Password	/native/DialupPassword	String Dialup password
Authentication	/native/DialupAuthentication	Dialup authentication. **PAP** is the default setting. Valid values are as follows.
		PAP CHAP

Label name	Internet Appliance keyword	Valid values
Dialup Phone Number Prefix	/native/DialupPhoneNumberPrefix	Prefix to use before the phone number when dialing. For example, 9
Modem Speaker Volume	/native/DialupModemVolume	Integer **1** is the default setting. Valid range is from 0 to 3.
Connect Timeout	/native/DialupConnectTimeout	Integer. Timeout in seconds for achieving a connection once the number is dialed. **30** is the default (seconds).
Retry	/native/DialupRetries	Integer. Number of dial retries, for situations such as a busy signal. **2** is the default.
Idle Timeout	/native/DialupIdleTimeout	Integer Disconnect after specified number of minutes of idle time. For example, 5
Dial Type	/native/DialupDialType	**Tone** is the default. Valid values are as follows.
		Tone Puls
Modem Init String	/native/DialupModemInitString	AT command set modem init string. For example, ATZ
Login Script	/native/DialupLoginScript	String Script string consisting of chat expect-send pairs. This in used only when Authentication is Manual.
DNS Server 1	/native/DialupDNSServer1	DNS server address (1st search). For example, 1.2.3.1
DNS Server 2	/native/DialupDNSServer2	DNS server address (2nd search). For example, 1.2.3.2

Label name	Internet Appliance keyword	Valid values
DNS Server 3	/native/DialupDNSServer3	DNS server address (3rd search). For example, 1.2.3.3

A.4.9 Ethernet tab

Table 56 shows the parameters in the Ethernet tab.

Table 56. IAD plug-in: Parameters in the Ethernet tab

Label name	Internet Appliance keyword	Valid values
Connection Type	/native/EthernetConnectonType	Ethernet connection type. **DHCP** is the default setting. Valid values are as follows.
		DHCP Static
DHCP Timeout	/native/EthernetDHCPTimeout	Integer Number of seconds to wait for DHCP server before timing out. **15** is the default (seconds).
IP Address	/native/EthernetIPAddress	Ethernet static IP address. For example, 5.2.8.110
Subnet Mask	/native/EthernetSubnetMask	Ethernet static subnet mask. For example, 255.255.255.0
Gateway	/native/EthernetGateway	Ethernet static gateway address. For example, 5.2.8.1
DNS Server 1	/native/EthernetDNSServer1	Ethernet static DNS server address (1st search). For example, 5.2.8.250
DNS Server 2	/native/EthernetDNSServer2	Ethernet static DNS server address (2nd search). For example, 5.2.8.251
DNS Server 3	/native/EthernetDNSServer3	Ethernet static DNS server address (3rd search). For example, 5.2.8.252
Domain	/native/EthernetDomain	Ethernet static domain. For example, dept.company.com

A.4.10 Syslog tab

Table 57 lists the parameters in the Syslog tab.

Table 57. IAD plug-in: Parameters in the Syslog tab

Label name	Internet Appliance keyword	Valid values
Syslog Severity	native/SyslogSeverity	Integer System log severity. **5** is the default setting. Valid range is from 0 to 10.

A.4.11 Email tab

Table 58 shows the parameters in the Email tab.

Table 58. IAD plug-in: Parameters in the Email tab

Label name	Internet Appliance keyword	Valid values
Mail Server	/shell/defaultSetting/Email /Mail Server	IMAP4 or POP3 e-mail server address. For example, email.company.com
Mail Server Protocol	/shell/defaultSetting/Email /Mail Server Protocol	IMAP4 is the default setting. Valid values are as follows.
		IMAP4 POP3
SMTP Server	/shell/defaultSetting/Email /SMTP Server	For example, smtp.company.com
Saved Folder Name	/shell/defaultSetting/Email /Saved Folder Name	Name of the server-side e-mail folder. For example, /my_mail
Check Mail Interval	/shell/defaultSetting/Email /Check Mail Interval	Integer. Interval in minutes to check for new e-mail. For example, 30 (minutes)
Email Massage Size Limit	/shell/defaultSetting/Email /Message Size Limit	Maximum size for a single e-mail message in KB. For example, 500 (KB)
Drafts Size Limit	/shell/defaultSetting/Email /Drafts Size Limit	Maximum size of the local draft folder in KB Draft folder is kept locally on the Internet appliance.For example, 200 (KB)

Label name	Internet Appliance keyword	Valid values
Address Book Size Limit	/shell/defaultSetting/Email /Address Book Limit	Maximum size of the local address book in KB. For example, 100 (KB)
Userid	/shell/defaultSetting/Email /Userid	String. E-mail user ID
Password	/shell/defaultSetting/Email /Password	String. E-mail password
Full Name	/shell/defaultSetting/Email /Full Name	It will appear in the e-mail as the user's full name. For example, Chris Jones
Email Address	/shell/defaultSetting/Email /Email Address	User's e-mail address. For example, user@e-mail_server

A.4.12 Connection Manager tab

Table 59 lists the parameters in the Connection Manager tab.

Table 59. IAD plug-in: Parameters in the Connection Manager tab

Label name	Internet Appliance keyword	Valid values
Interfaces	/native/ConnectionInterfaces	An ordered list of the interface names the connection manager is to attempt to connect to. Use a space to separate list members. For example, lan.eth0 v90.ppp0

A.4.13 RAM drive tab

Table 60 shows parameter in the RAM drive tab.

Table 60. IAD plug-in: Parameters in the RAM drive tab

Label name	Internet Appliance keyword	Valid values
dev-f RAM Size	/native/ramfssize	Integer Size in MB of the file system created in memory. **2** is the default. Valid range is from 1 to 16

Appendix B. Special notices

This publication is intended to help people who need to understand the concepts and implementations of Tivoli Personalized Services Manager Device Manager Version 1.1, and who need to develop their own device plug-ins to manage their special devices. The information in this publication is not intended as the specification of any programming interfaces that are provided by Tivoli Personalized Services Manager Device Manager Version 1.1 and IBM WebSphere Everyplace Suite Version 1.1. See the PUBLICATIONS section of the IBM Programming Announcement for Tivoli Personalized Services Manager Device Manager Version 1.1 and IBM WebSphere Everyplace Suite Version 1.1 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, North Castle Drive, Armonk, NY 10504-1785.

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 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 pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

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

e (logo)®
IBM ®

AIX
DB2
Lotus
MQSeries
NetVista
SecureWay
WebSphere

Redbooks
Redbooks Logo

AS/400
DB2 Universal Database
Lotus Notes
Netfinity
RS/6000
System/390

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Appendix C. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 IBM Redbooks

For information on ordering these publications see "How to get IBM Redbooks" on page 263.

- *An Introduction to IBM WebSphere Everyplace Suite Version 1.1,* SG24-5995
- *Database Performance on AIX in DB2 UDB and Oracle Environments*, SG24-5511
- *IBM HTTP Server Powered by Apache on RS/6000*, SG24-5132
- *WebSphere Application Servers: Standard and Advanced Editions*, SG24-5460
- *IBM WebSphere Performance Pack: Load Balancing·with IBM SecureWay Network Dispatcher*, SG24-5858
- *IBM Network Dispatcher User's Guide (Version 3.0 for Multiplatforms)*, GC31-8496

C.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at **ibm.com**/redbooks for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	**Collection Kit Number**
IBM System/390 Redbooks Collection	SK2T-2177
IBM Networking Redbooks Collection	SK2T-6022
IBM Transaction Processing and Data Management Redbooks Collection	SK2T-8038
IBM Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
IBM AS/400 Redbooks Collection	SK2T-2849
IBM Netfinity Hardware and Software Redbooks Collection	SK2T-8046
IBM RS/6000 Redbooks Collection	SK2T-8043
IBM Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

C.3 Other resources

These publications are also relevant as further information sources:

- *HACMP for AIX 4.4 Planning Guide*, SC23-4277

The following publications mentioned in this redbook are Product Documentation, which can be obtained in Tivoli product CD-ROMs or IBM WebSphere Everyplace Suite product CD-ROMs:

- *WebSphere Everyplace Suite Getting Started v1.1*
- *Tivoli Internet and Personalized Services Manager Documentation: TPSM Overview*
- *Tivoli Internet and Personalized Services Manager Documentation: Planning and Installation*
- *Tivoli Internet and Personalized Services Manager Documentation: Administration*
- *Tivoli Internet and Personalized Services Manager Documentation: Director Guide*
- *Tivoli Internet and Personalized Services Manager Documentation: Programmer's Guide*
- *Tivoli Personalized Services Manager Device Manager: Planning and Installation*
- *Tivoli Personalized Services Manager Device Manager: Administration*
- *Tivoli Personalized Services Manager Device Manager: Developer's Guide*
- *Tivoli Personalized Services Manager Device Manager: PalmOS Plug-in Notes*
- *Tivoli Personalized Services Manager Device Manager: NetVista Internet Appliance Plug-in Notes*
- *Tivoli Personalized Services Manager Device Manager: Aero 8000 Plug-in Notes*
- *Tivoli Personalized Services Manager Device Manager: Windows CE Plug-in Notes*

C.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- `http://www.jp.ibm.com/ise/english/ecomp.htm`
  ISE Co., Ltd

- http://www.palm.com
  Palm Computing
- http://www.compaq.com/products/handhelds/8000
  Compaq Aero 8000
- http://www.pc.ibm.com/us/netvista/index.html
  IBM NetVista
- http://www.ibm.com/software/network/dispatcher
  IBM Network Dispatcher
- http://www.ibm.com/security/library/
  IBM Security general information
- http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/hacmp_index.html#V44
  IBM HACMP Version 4.4
- http://www.ibm.com/pvc/tech/library.shtml
  IBM Pervasive technical library
- http://www.sun.com/software/solaris/java/download.html
  Sun JDK download site
- http://www.ibm.com/software/webservers/httpservers
  IBM HTTPServer
- http://www.ibm.com/software/webservers/appserv
  IBM WebServer Apach server
- http://www.ibm.com/software/webservers/edgeserver
  IBM Network Dispatcher User's Guide
- http://java.sun.com
  Sun Java Technology
- http://technet.oracle.com/software/tech/java/sqlj_jdbc/software_index.htm
  JDBC driver for Oracle database (Registration in free is required.)
- http://java.sun.com/products/jdbc
  Sun JDBC driver general information
- http://sunsolve.sun.com
  Sun Solaris 7 PTF Download Site

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** **ibm.com**/redbooks

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  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.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax 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.

Glossary

Aero 8000 The Compaq Aero 8000 Handheld PC, which is a kind of Windows CE machine. It uses Hitachi SH4 processor and has a 10-inch color SVGA screen with 800 x 600 resolution, 256 colors, and a 0.24 pixel pitch.

AIX Advanced Interactive eXecutive. An IBM industrial-strength version of UNIX.

API Application program interface.

ASP Application Service Provider. An Internet service provider that offers application services.

business object An object containing business methods (logic) and state that is intended for use within business applications. Business objects are Managed Objects. In some contexts, the term “business object” in this book is used to refer to a business object class. It may also be used to refer to a composition of business object classes.

Challenge Handshake Authentication Protocol (CHAP) A type of authentication in which the authentication agent (typically a network server) sends the client program a key to be used to encrypt the username and password.

CHAP See *Challenge Handshake Authentication*.

conduit A component of Palm Desktop software. A conduit specifies how to transfer and translate data between your handheld and your computer for a specific handheld application or database during a Hot Sync operation.

cradle The docking station for PDA devices to connect to the desktop PC. Generally, they also have the function to charge the device.

DBCS Double byte character set.

DDL Data Definition Language.

deal Deals represent the service plan that is being subscribed to.

Device management API An API (package com.tivoli.dms.dmapi) that defines the programming interface for manipulating the device- and job-related data resources stored in the Device Manager database.

Device management server API An API (package com.tivoli.dms.dmserver) that defines the programming interface between the device management server servlet and the device plug-ins.

Device Manager console A graphical user interface, written by Java, for administering device management operations.

DHCP See *Dynamic Host Configuration Protocol*.

DM Device Manager.

DM console see *Device Manager console*.

DMD Device Management Dispatcher.

DMS servlet device management server servlet.

DNS See *Domain Name System*.

Domain Name System (DNS) A function to associate names and addresses on Internet domain servers.

Dynamic Host Configuration Protocol (DHCP) A protocol for assigning dynamic IP addresses to devices on a network.

e-business A term used by IBM to describe the use of Internet technologies to transform business processes. In practice, this means using Internet clients, such as Web browsers, as front ends for applications that access back-end legacy systems to allow greater access. See `http://www.software.ibm.com/ebusiness` for more information.

eXtensible Markup Language (XML) This markup language, a streamlined version of SGML, is regulated by WC3 (the World Wide Web Consortium). Can create more advanced links than HTML.

file package Describes which files and directories to distribute and how to distribute them.

file package definition file An ASCII file that identifies the contents and characteristics of a file package.

File Transfer Protocol (FTP) The protocol used for sending files over the network.

FTP See *File Transfer Protocol*.

GUI Graphical User Interface.

HACMP High Availability Cluster Multi-Processing.

HotSync A Palm function to do two-way synchronization of records between your handheld and your computer. Changes that you make on your handheld or your computer are updated on both platforms after a HotSync operation. The HotSync operation only synchronizes the changed portions of data, reducing HotSync time.

HTTP See *Hyper Text Transmission Protocol*.

Hyper Text Transmission Protocol (HTTP) The communications protocol used to connect to servers on the World Wide Web. Its primary function is to establish a connection with a Web server and transmit HTML pages to the client browser.

IBM International Business Machines.

IMAP See *Internet Message Access Protocol.*

Integrated Services Digital Network (ISDN) An international communications standard for sending voice, video, and data over digital telephone lines or normal telephone wires.

Internet Message Access Protocol (IMAP) A protocol for retrieving e-mail messages.

Internet screenphone A telephone-like appliance with a built-in display screen that may be used to call up Internet sites quickly.

Internet Service Provider (ISP) Offers services on the Internet, such as connection services.

ISDN See *Integrated Services Digital Network*.

ISP See *Internet Service Provider*.

iTk Tivoli Internet Services Manager Integration Toolkits.

Java A high-level programming language developed in 1991 by Sun Microsystems that works on virtually any computer. Unlike HTML, which is a document display format that is continually improved to make it do more, Java is a full-blown programming language like C and C++. It allows for the creation of sophisticated client/server applications to be developed for the Web and for intranets.

Java Database Connectivity (JDBC) A Java API that allows Java programs to communicate with different database management systems in a platform-independent manner. Database vendors provide JDBC drivers for their platforms that implement the API for their database, allowing the Java developer to write applications to a consistent API no matter which database is used.

Java Development Kit (JDK) A software development kit (SDK) for producing Java programs.

JDBC See *Java Database Connectivity*.

JDK See *Java Development Kit*.

kiosk services Solutions for business center service providers such as hotel chains, airports, and office service franchises that offer document services (storing, reproduction, and distribution) through shared access devices in public spaces.

LAN See *Local Area Network*.

LDAP See *Lightweight Directory Access Protocol*.

Lightweight Directory Access Protocol (LDAP) A set of protocols for accessing information directories. LDAP is based on the X.500 protocol, but supports TCP/IP, which is necessary for Internet access. Because it's a simpler version of X.500, LDAP is sometimes called X.500-lite.

Local Area Network (LAN) A computer network that spans a relatively small area, such as a building or a group of buildings.

meta file definition file An ASCII file that provides software distribution properties for the job, also has a numbered application stanza, [Applicationn], for each application package in the software distribution job.

NetVista The IBM NetVista™ systems include all-in-one, legacy-free, traditional desktops, and thin clients to provide a full range of computing solutions for your business or home.

Palm OS platform A popular operating system for Palm and other handheld devices, designed by Palm specifically for information management.

Palm Pilot One of the most popular brands of PDAs, which uses Palm OS provided by Palm, Inc. Nowadays, 3Com, IBM, Sony, and some other vendors sell their original Palm Pilot devices.

Palm OS resource database Commonly referred to as a PRC (Palm Resource). A PRC can exist as a file on the host computer (that is, a PC or a Macintosh) and is commonly referred to as a PRC file. A PRC contains database header information as well as individual resource records in the database itself.

Palmscape Palmscape is a Web browser for PDA. Palm version of Palmscape works on the Palm OS 3.0 and above.

PAP See *Password Authentication Protocol*.

Password Authentication Protocol (PAP) The most basic form of authentication, in which a user's name and password are transmitted over a network and compared to a table of name-password pairs.

PDA See *Personal Digital Assistant*.

PDB PDB files contain database records that are used by Palm OS to store the application data. They have features unique to the Palm OS.

Personal Digital Assistant (PDA) A small handheld device that offers functions such as address storage, calendar, and e-mail. This mobile device can be synchronized with desktop PCs.

Point-to-Point Protocol (PPP) A data link protocol that provides dial-up access over several lines.

POP See *Post Office Protocol*.

Portal Toolkit (pTk) A toolkit which features a Multi-device portal engine which supports web devices such as Win CE based PDAs, screenphones, WAP devices, as well as PCs. It also features a User Preference API, which allows user and users group profiling.

Post Office Protocol (POP) A protocol used to retrieve e-mail from a mail server.

PPP See *Point-to-Point Protocol*.

PRC Palm Resource. See *Palm OS resource database*.

proxy A proxy server is configured to manage network traffic and to protect your network.

PSTN See *Public Switched Telephone Network*.

PTF Problem temporary fix.

pTk See *Portal Toolkit*.

Public Switched Telephone Network (PSTN) The international telephone system based on copper wires carrying analog voice data. This is in contrast to newer telephone networks based on digital technologies, such as ISDN and FDDI.

RADIUS Remote Access Dial-In User Service.

RAM disk Refers to RAM that has been configured to simulate a disk drive. You can access files on a RAM disk as you would access files on a real disk.

RAS Short for Reliability, Availability, Serviceability. See *Remote Access Services*.

Realm Realms are a division of the subscriber name-space that allow the ISP to divide its Internet provisioning services, effectively creating a set of “Virtual ISPs”.

Remote Access Services (RAS) A feature built into Windows NT that enables users to log into an NT-based LAN using a modem, X.25 connection or WAN link.

screenphone See *Internet screenphone*.

servlets Java classes that run on Web servers to provide dynamic HTML content to clients. They take as input the HTTP request from the client and output dynamically generated HTML. For more information on servlets, see `http://www.software.ibm.com/ebusiness/pm.html#Servlets`

Simple Mail Transfer Protocol (SMTP) A protocol for sending e-mail messages between servers.

SM See *Subscription Manager*.

smart phone Enables connections to the Internet or Lotus Notes network so that users can access e-mail, faxes, voice mail, Web pages and other files. It also has the potential to connect to a speech recognition server by phone, which would allow a user to dictate notes or responses and then see the recognized text as it is returned from the server.

SMTP See *Simple Mail Transfer Protocol*.

Solaris A UNIX-based operating environment developed by Sun Microsystems, originally developed to run on Sun's SPARC workstations, it now runs on many workstations from other vendors. Solaris includes the SunOS operating system and a windowing system (either Open Windows or CDE).

SQL See *Structured Query Language*.

SSL Secure Sockets Layer. A protocol for transmitting private documents securely via the Internet.

Structured Query Language (SQL) A standard set of statements used to manage information stored in a database. By using these statements, users can add, delete, or update information in a table, request information through a query, and display the result in a report.

Subscription Manager (SM) A feature of the *Tivoli Internet and Personalized Services Manager: Programmer's Guide*, which has functions such as enrollment, access control, and personalization.

TCP/IP Transmission Control Protocol/Internet Protocol.

TISM Tivoli Internet Services Manager.

TPSM Tivoli Personalized Services Manager.

transcoding The operation of changing data from one format to another, such as XML to HTML, so that the output will be displayed in a manner appropriate to the device.

TSM Tivoli Subscriber Manager.

UDB DB2 Universal Database from IBM.

URL Universal Resource Locator.

WAN Wide Area Network.

WAP See *Wireless Application Protocol*.

WAS IBM WebSphere Application Server.

WE See *WebSphere Everyplace Suite*.

WebSphere Everyplace Suite (WES) IBM WebSphere Everyplace Suite is the solution for pervasive computing that connects any device to any data, anywhere, anytime. It supports the following features: Connectivity, Content Adaptation, Optimization, Security, Management Services, and Services.

WebSphere Transcoding Publisher (WTP) A network software that modifies content presented to users based on the information associated with the request, such as device constraints, network constraints, user preferences, and organization policies. Transforming content can reduce or eliminate the need to maintain multiple versions of data or applications for different device types and network service levels.

Windows CE A version of the Windows operating system designed for small devices such as personal digital assistants (PDAs) (or handheld PCs in the Microsoft vernacular). The Windows CE graphical user interface (GUI) is similar to Windows 95 so devices running Windows CE should be easy to operate for anyone that is familiar with Windows 95.

Wireless Application Protocol (WAP) A protocol to transfer content to and from wireless devices.

Wireless Markup Language (WML) A language to present content on wireless devices.

WML See *Wireless Markup Language*.

WTE Web Traffic Express.

WTP See *WebSphere Transcoding Publisher*.

XML See *eXtensible Markup Language*.

Index

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 845 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-6027-00 Tivoli Personalized Services Manager Device Manager 1.1: Pervasive Device Management
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	 O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

Tivoli Personalized Services Manager Device Manager 1.1

(0.5” spine)
0.475”<->0.875”

Tivoli Personalized Services Manager Device Manager 1.1

Pervasive Device Management

Understand how to use Device Manager for pervasive device management

Install, tailor, and configure the Device Manager server

Learn useful information to help you develop plug-ins

Device Manager Version 1.1 is software that functions as a part of Tivoli Personalized Services Manager Version 1.1, or IBM WebSphere Everyplace Suite Version 1.1. It enables pervasive devices to easily access a set of network-based services, as well as Internet-based e-commerce services.

This IBM Redbook will assist developers and system architects who are involved in building pervasive device management solutions that use Device Manager Version 1.1. It explains how Device Manager Version 1.1 can be used to manage devices in the growing pervasive computing world. Pervasive devices are typically small, resource-limited, and not perceived as computers; however, growth of these information appliances is increasing at an amazing rate.

In this redbook, you will find information that to help you plan, tailor, and configure the Device Manager to successfully implement solutions that an e-business must address to access Internet-based services from pervasive devices such as Palm devices, screenphones, and Wireless Application Protocol (WAP) devices.

This redbook also looks at software distribution and device configuration examples, introduction of supported APIs, and an explanation of device plug-ins. It helps you to plan and make your own device plug-in when you want to manage special devices not provided with Device Manager Version 1.1.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6027-00 ISBN 0738421162