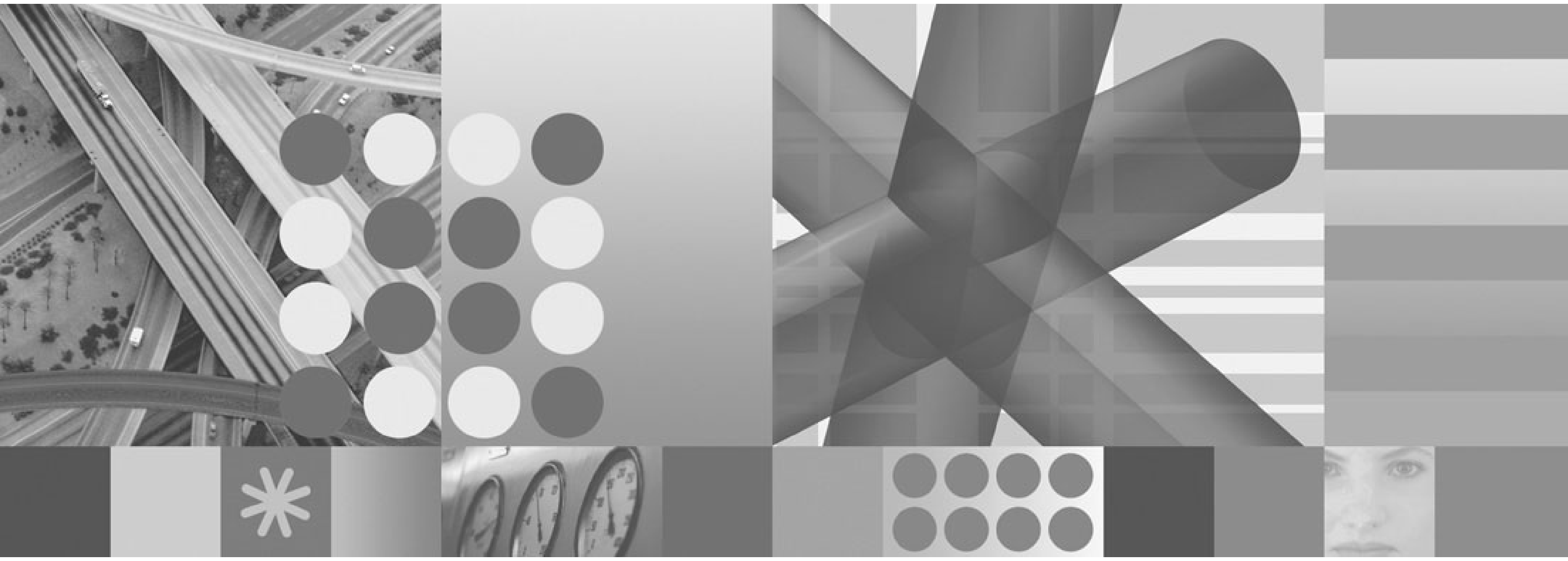
Tivoli® Directory Integrator

IBM Tivoli Directory Integrator 6.1.1: Administrator Guide

SC32-2567-01

Tivoli® Directory Integrator



IBM Tivoli Directory Integrator 6.1.1: Administrator Guide

SC32-2567-01

Note

Note: Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 185.

Second Edition (February 2007)

This edition applies to version 6.1.1 of the IBM Tivoli Directory Integrator and to all subsequent releases and modifications until otherwise indicated in new editions.

Preface

This document contains the information that you need to develop solutions using components that are part of the IBM® Tivoli® Directory Integrator.

Who should read this book

This book is intended for those responsible for the development, installation and administration of solutions with the IBM Tivoli Directory Integrator.

Tivoli Directory Integrator components are designed for network administrators who are responsible for maintaining user directories and other resources. This document assumes that you have practical experience installing and using both IBM Tivoli Directory Integrator and

The reader should be familiar with the concepts and the administration of the systems that the developed solution will connect to. Depending on the solution, these could include, but are not limited to, one or more of the following products, systems and concepts:

- IBM Directory Server
- IBM Tivoli Identity Manager
- IBM Java™ Runtime Environment (JRE) or Sun Java Runtime Environment
- Microsoft® Active Directory
- PC and UNIX® operating systems
- Security management
- Internet protocols, including HTTP, HTTPS and TCP/IP
- Lightweight Directory Access Protocol (LDAP) and directory services
- A supported user registry
- Authentication and authorization
- SAP R/3.

Publications

Read the descriptions of the IBM Tivoli Directory Integrator library and the related publications to determine which publications you might find helpful. After you determine the publications you need, refer to the instructions for accessing publications online.

IBM Tivoli Directory Integrator library

The publications in the IBM Tivoli Directory Integrator library are:

IBM Tivoli Directory Integrator 6.1.1: Getting Started
A brief tutorial and introduction to IBM Tivoli Directory Integrator 6.1.1.

IBM Tivoli Directory Integrator 6.1.1: Administrator Guide
Includes complete information for installing the IBM Tivoli Directory Integrator. Includes information about migrating from a previous version of IBM Tivoli Directory Integrator. Includes information about configuring the logging functionality of IBM Tivoli Directory Integrator. Also includes information about the security model underlying the Remote Server API.

IBM Tivoli Directory Integrator 6.1.1: Users Guide
Contains information about using the IBM Tivoli Directory Integrator 6.1.1 tool. Contains instructions for designing solutions using the IBM Tivoli Directory Integrator tool (**ibmditk**) or running the ready-made solutions from the command line (**ibmdisrv**). Also provides information

about interfaces, concepts and AssemblyLine/EventHandler creation and management. Includes examples to create interaction and hands-on learning of IBM Tivoli Directory Integrator 6.1.1.

IBM Tivoli Directory Integrator 6.1.1: Reference Guide
Contains detailed information about the individual components of IBM Tivoli Directory Integrator 6.1.1 AssemblyLine (Connectors, EventHandlers, Parsers, Plug-ins, and so forth).

IBM Tivoli Directory Integrator 6.1.1: Problem Determination Guide
Provides information about IBM Tivoli Directory Integrator 6.1.1 tools, resources, and techniques that can aid in the identification and resolution of problems.

IBM Tivoli Directory Integrator 6.1.1: Messages Guide
Provides a list of all informational, warning and error messages associated with the IBM Tivoli Directory Integrator 6.1.1.

IBM Tivoli Directory Integrator 6.1.1: Password Synchronization Plug-ins Guide
Includes complete information for installing and configuring each of the five IBM Password Synchronization Plug-ins: Windows Password Synchronizer, Sun ONE Directory Server Password Synchronizer, IBM Directory Server Password Synchronizer, Domino Password Synchronizer and Password Synchronizer for UNIX and Linux®. Also provides configuration instructions for the LDAP Password Store and MQe Password Store.

IBM Tivoli Directory Integrator 6.1.1: Release Notes
Describes new features and late-breaking information about IBM Tivoli Directory Integrator 6.1.1 that did not get included in the documentation.

Related publications

Information related to the IBM Tivoli Directory Integrator is available in the following publications:

- IBM Tivoli Directory Integrator 6.1.1 uses the JNDI client from Sun Microsystems. For information about the JNDI client, refer to the *Java Naming and Directory Interface™ 1.2.1 Specification* on the Sun Microsystems Web site at http://java.sun.com/products/jndi/1.2/javadoc/index.html.
- The Tivoli Software Library provides a variety of Tivoli publications such as white papers, datasheets, demonstrations, redbooks, and announcement letters. The Tivoli Software Library is available on the Web at: http://www.ibm.com/software/tivoli/library/
- The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available on the World-Wide Web, in English only, athttp://publib.boulder.ibm.com/tividd/glossary/tivoliglossarymst.htm

Accessing publications online

The publications for this product are available online in Portable Document Format (PDF) or Hypertext Markup Language (HTML) format, or both in the Tivoli software library: http://www.ibm.com/software/tivoli/library.

To locate product publications in the library, click the **Product manuals** link on the left side of the Library page. Then, locate and click the name of the product on the Tivoli software information center page.

Information is organized by product and includes READMEs, installation guides, user's guides, administrator's guides, and developer's references as necessary.

Note: To ensure proper printing of PDF publications, select the **Fit to page** check box in the Adobe Acrobat Print window (which is available when you click **File->Print**).

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully. With TDI 6.1.1, you can use assistive technologies to hear and navigate the interface. After installation you also can use the keyboard instead of the mouse to operate all features of the graphical user interface.

Accessibility features

The following list includes the major accessibility features in TDI 6.1.1:

- Supports keyboard-only operation.
- Supports interfaces commonly used by screen readers.
- Discerns keys as tactually separate, and does not activate keys just by touching them.
- Avoids the use of color as the only way to communicate status and information.
- Provides accessible documentation.

Keyboard navigation

This product uses standard MicrosoftWindows® navigation keys for common Windows actions such as access to the File menu, copy, paste, and delete. Actions that are unique to TDI use TDI keyboard shortcuts. Keyboard shortcuts have been provided wherever needed for all actions.

Interface Information

The following points include accessibility features of the TDI 6.1.1 user interface and documentation:

- Steps for changing fonts, colors, and contrast settings in the Config Editor (CE):
  1. Type **Alt-F** to access the CE **File** menu. Using the downward arrow, select **Edit Preferences** and press **Enter**.
  2. Under the **Appearance** tab, select **Theme** settings to change the font.
  3. Under **Theme Colors**, select the colors for the CE, and by selecting colors, you can also change the contrast.
- The TDI 6.1.1 Information Center and its related publications are accessibility-enabled for the JAWS screen reader and the IBM Home Page Reader. You can operate all documentation features using the keyboard instead of the mouse.

Vendor software

The IBM Tivoli Directory Integrator installer uses the FLEXnet Publisher Installation Module (FNPIM).

Related accessibility information

Visit the *IBM Accessibility Center* at http://www.ibm.com/able for more information about IBM's commitment to accessibility.

Contacting IBM Software support

Before contacting IBM Tivoli Software support with a problem, refer to IBM System Management and Tivoli software Web site at:

http://www.ibm.com/software/sysmgmt/products/support/

If you need additional help, contact software support by using the methods described in the *IBM Software Support Handbook* at the following Web site:

http://techsupport.services.ibm.com/guides/handbook.html

The guide provides the following information:

- Registration and eligibility requirements for receiving support
- Telephone numbers and e-mail addresses, depending on the country in which you are located
- A list of information you must gather before contacting customer support

Contents

Chapter 1. Introduction

For an overview of the general concepts of the IBM Tivoli Directory Integrator 6.1.1, refer to "IBM Tivoli Directory Integrator concepts," in *IBM Tivoli Directory Integrator 6.1.1: Users Guide*.

For more detailed information about IBM Tivoli Directory Integrator 6.1.1 concepts, see *IBM Tivoli Directory Integrator 6.1.1: Reference Guide*.

Chapter 2. Installation instructions for IBM Tivoli Directory Integrator

Before you install

Before you install, please read the following sections and make sure your system meets the minimum requirements.

Disk Space Requirements

The IBM Tivoli Directory Integrator 6.1.1 Solution Installer requires 450 MB of temporary disk space during installation, and additionally the following amount of diskspace for the TDI components that will remain on the box after installation:

Disk space requirements by platform for a Typical installation:

- Windows (32 and 64bit): 341 MB
- Linux (32 and 64bit): 413 MB
- AIX: 342 MB
- Solaris: 453 MB
- HPUX: 562 MB

Disk space requirements by platform for a Custom installation in which all components are selected:

- Windows (32 and 64bit): 564 MB
- Linux: (32 and 64bit) 643 MB
- AIX: 556 MB
- Solaris: 773 MB
- HPUX: 858 MB

The precise amount of required disk space depends on the components you choose to install. To calculate precisely the necessary disk space, add together the disk space requirements for each component you want to install. See “Components in IBM Tivoli Directory Integrator” on page 4 for the required disk space for each TDI component.

Memory Requirements

The IBM Tivoli Directory Integrator 6.1.1 Installer requires 512 MB of memory. The precise amount of required memory after installation depends on the components you choose to install.

To calculate the necessary memory requirements, add together the memory requirements for each component you want to install. See “Components in IBM Tivoli Directory Integrator” on page 4 for the memory requirements of each TDI component.

Memory requirements for a Typical installation: 484 MB

Memory requirements for a Custom installation with all components: 868 MB

Platform Requirements

See Chapter 3, “Supported platforms,” on page 41

Components in IBM Tivoli Directory Integrator

The following components are available and selectable for installation as part of IBM Tivoli Directory Integrator 6.1.1:

Runtime Server

A rules engine used to deploy and run TDI integration solutions.

- Disk space requirements: 25 MB
- Memory requirements: Each TDI server instance requires at least 256 MB. NOTE: More RAM may be required depending on the size and complexity of the solution being created.

Config Editor

A development environment for creating, debugging and enhancing TDI integration solutions. If this component is selected, the Runtime Server is also selected by default.

- Disk space requirements: 2.5 MB
- Memory requirements: 128 MB.

Javadocs

Full HTML documentation of TDI internals. Essential reference material for scripting in solutions, as well as for developing custom components.

- Disk space requirements: 59 MB
- Memory requirements: N/A

Examples

A series of short, illustrative example Configs that highlight specific TDI features or components.

- Disk space requirements: 2.5 MB
- Memory requirements: N/A

IEHS v3.11 (local help)

You can install an IBM Eclipse Help System locally as an alternative to using the global online help service. This option requires manual download and deployment of the TDI help files after installation.

Disk space requirements by platform:

- Windows (32 and 64bit): 21.2 MB
- Linux (32 and 64bit): 14.8 MB
- AIX: 14.7 MB
- Solaris: 14.8 MB
- HPUX: 14.7 MB

Memory requirements: 128 MB. 256 MB or more is recommended.

Note: You need to increase memory according to the size of the documentation plug-ins. For example, if the size of the documentation is 100 MB, add at least 80 MB of additional RAM.

If your platform meets these requirements, you can proceed with the download and install instructions documented in "Installing local Help files" on page 16.

AMC: Administration and Monitoring Console

A browser-based application for monitoring and managing running TDI Servers.

- Disk space requirements: 74 MB
- Memory requirements: 128 MB

Embedded version of WebSphere Express v6.0.2

A lightweight application server you can install as an alternative to deploying Administration Monitoring Console (AMC) on an existing WAS installation. If this component is selected AMC is also selected by default.

Disk space requirements by platform:

- Windows (32 and 64bit): 172 MB
- Linux (32 and 64bit): 141 MB
- AIX: 125 MB
- Solaris: 231 MB
- HPUX: 207 MB

Memory requirements: 128 MB

Note: Embedded version of WebSphere Express v6.0.2 requires the use of ports. The ports for the embedded version of WAS shipped with TDI are:

- HTTP – 13100
- HTTPS – 13101
- RMI – 13102
- SOAP - 13103

Additional components automatically installed that are not selectable:

JRE (Java Runtime Environment) 5.0 SR3

A subset of the Java Development Kit (JDK) that contains the core executables and files that constitute the standard Java platform. The JRE includes the Java Virtual Machine (JVM), core classes, and supporting files.

Disk space requirements by platform:

- Windows: 60 MB
- Linux: 60 MB
- AIX: 60 MB
- Solaris: 120 MB
- HP: 100 MB

Memory requirements: N/A

Note: This package is installed automatically (one copy only) when any of the following components are installed:

- Runtime Server
- Config Editor
- IEHS v3.11

The JRE used for any of the installed TDI packages is independent of any system-wide JRE or JDK you may have installed on your system.

TDI 6.1.1 License Package

The license files for IBM Tivoli Directory Integrator 6.1.1.

Disk space requirements: 7KB

Memory requirements: N/A

Note: This package is installed automatically (one time only) when any other selectable TDI component is installed.

TDI 6.1.1 Uninstaller

The uninstaller program for TDI 6.1.1.

Disk space requirements by platform:

- Windows: 73 MB
- Linux: 89 MB
- AIX: 73 MB
- Solaris: 98 MB
- HPUX: 167 MB

Memory requirements: N/A

Note: This package is installed automatically (one time only) when any other selectable TDI component is installed.

SI (Solution Install) 1.2.1 FP 18

The IBM Tivoli Directory Integrator 6.1.1 Installer is built on top of the in IBM Solution Install (SI) technology. SI is a platform neutral installation registry component. During installation of TDI, IBM Solution install 1.2.1 FP 18 is installed and is used by other SI based installers in the future.

Disk space requirements by platform:

- Windows: 101 MB
- Linux: 157 MB
- AIX: 102 MB
- Solaris: 129 MB
- HPUX: 188 MB

Memory requirements: 100 MB

Note: This package is installed automatically (one time only) when any other selectable TDI component is installed. Also, this component requires the use of port 4130.

DEUI (Deployment Engine Update Installer) 1.2.1

The Deployment Engine Update Installer is an application that will be used to install maintenance fixes for TDI 6.1.1 and other products whose installers are SI based.

Disk space requirements:17 MB

Memory requirements: N/A

Other requirements

Solution Install Considerations

You do not need to take any action in regard to the Solution Installer, but be aware of the following issues before you run the TDI 6.1.1 Installer:

- SI will be installed during the initial portion of the installer if it does not exists or if your box has an older version of SI currently installed. SI will remain on your box even if you cancel out of the installer.
- Solution Install will not be removed from your box when IBM Tivoli Directory Integrator 6.1.1 is uninstalled because it is an install registry that is shared by other products.
- SI as an installation registry server on your machine will require the use of port 4130 to receive requests. If you have a firewall on your machine you will need enable localhost traffic on port 4130. If your firewall asks you to allow the TDI 6.1.1 installer/uninstaller to contact an application running on port 4130 on localhost, you must allow it or the installer/uninstaller will fail.

Note: The IBM Tivoli Directory Integrator 6.1.1 installer/uninstaller will only be making a connection to your local box and nothing external.

Root or Administrator Privileges

On Windows platforms, the installer (by default) requires that the ID you use to install TDI be the Administrator ID or a member of the Administrators group. On UNIX platforms, the installer (by default) requires that you run as root, or are able to execute the `sudo` command. The installer will fail if the user ID used to install TDI does not have these privileges.

We recommend that TDI is installed by an Administrator or root id, for respectively Windows and UNIX platforms. However, in order for a non-admin user to install, you must use the option `-siinstall true` on the command line. If you do not have admin privileges, you cannot uninstall an admin installed TDI.

If you as a non-admin user are installing TDI, and SI has not been installed globally (by an Administrator or root) then a local copy of SI will be created in your home directory. If SI has been installed globally (by an admin) then the non-admin install will use the global SI.

Notes:

1. You must have the proper authority to write to the specified install directory, if you use `-siinstall true`.
2. IBM Tivoli Directory Integrator 6.1.1 will not be seen or available to other users on the machine unless a global SI is already installed on the Operating System.
3. On AIX and i5/OS, root or Administrator privileges are required: non-Administrative installs are not supported.

Installation location

A number of dialogs during the installation process let you choose where IBM Tivoli Directory Integrator 6.1.1 will be installed. Generally, this can be anywhere on a file system for which you have write access; and if you choose the installation directory also to be the Solution Directory, then during execution of the Server the identity under which the Server runs will need to have write access to that location too.

The Solution Installer does not allow you to install any product at the root level. For Linux/UNIX (all flavors), this means you cannot install TDI in the "/" directory; and on Windows systems you cannot install in the "\" directory on *any* drive, that is, not in "C:\", but not on "D:\" or "E:\" either or the top level of any other letter drive.

In addition, we recommend that the install path on a Linux/UNIX system not contain the space (" ") character, as this is likely to cause problems. In fact, if you choose to install AMC and/or Embedded WAS Express the installation will certainly fail if the space character is present anywhere in the path.

Installing IBM Tivoli Directory Integrator

IBM Tivoli Directory Integrator 6.1.1 installer allows you to install all TDI components or only those components you need. The following sections contain information about installing TDI.

Notes:

1. On i5/OS, in order to extract the TDI 6.1.1 installer from a TAR image, you need to set environment variable ″QIBM_CCSID″ to 819. that is, execute command

   ```
   export QIBM_CCSID=819
   ```

   before un-tarring the TDI installer TAR image.
2. On i5/OS, there is no Launchpad; therefore you will need to launch the installer directly as indicated below. The installer can be found in the TDI_INST directory.

Launching the appropriate installer

You can launch the IBM Tivoli Directory Integrator 6.1.1 Installer by using one of the following methods:

Launch the installer from the Launchpad

The TDI Launchpad provides key getting started installation information and links to more detailed information on various installation, migration, and post install topics.

Note: The Launchpad is not available on z/OS and i5/OS.

In addition, it provides a mechanism to launch the TDI installer. To start Launchpad, type the following at the command prompt:

1. Open the TDI Launchpad by typing the following at the command prompt:
   - For Windows platforms, type:

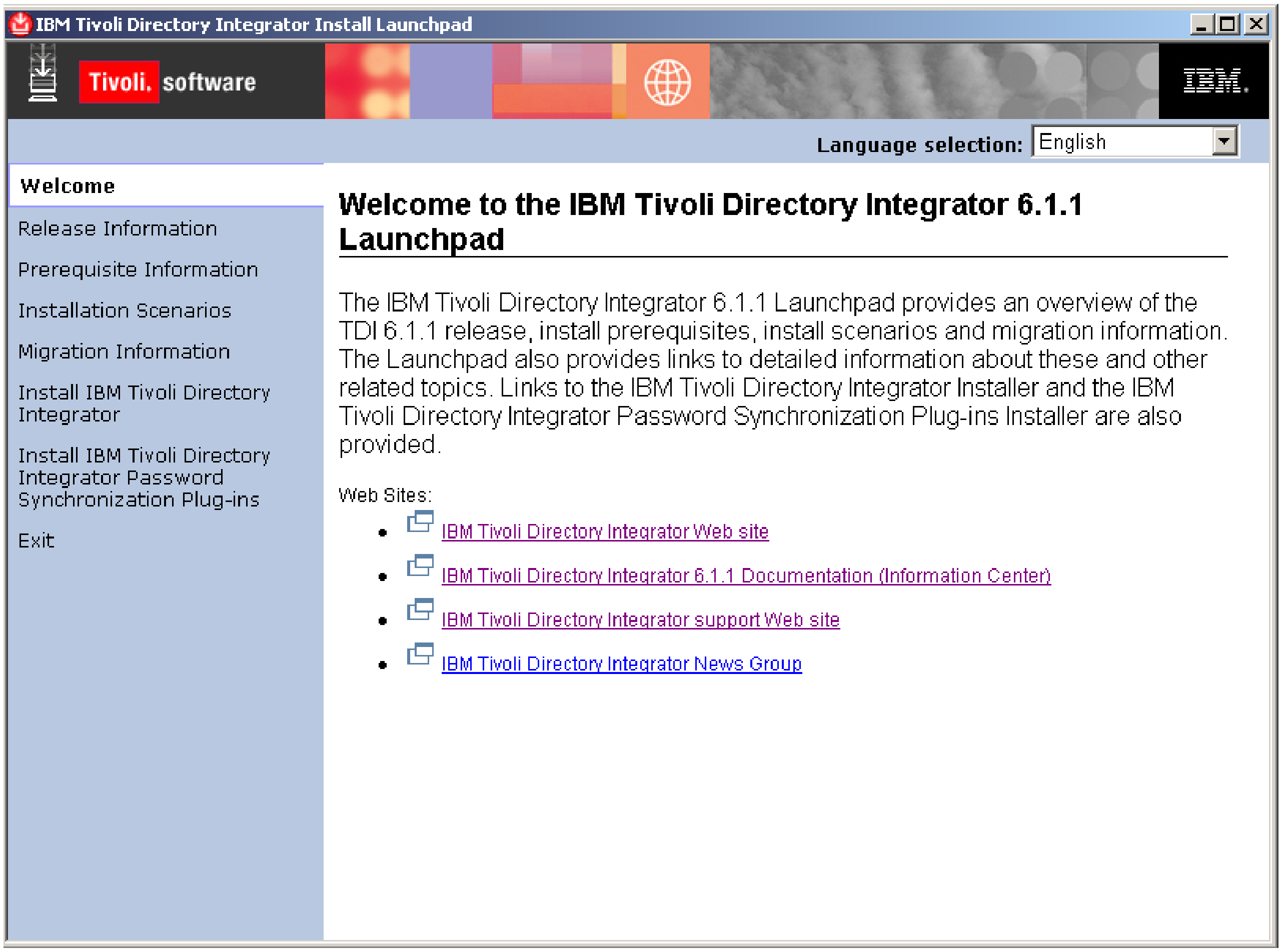
     ```
     Launchpad.bat
     ```
   - For all other platforms, type:

     ```
     Launchpad.sh
     ```

   The menu on the left of the Launchpad allows you to navigate the Launchpad panels. Click a menu item to view information about it. The following menu items are available:

   Welcome

   The Welcome panel contains links to
   - TDI Website
   - 6.1.1 Documentation
   - Support Web site
   - TDI newsgroup


IBM Tivoli Directory Integrator Install Launchpad
Tivoli. software
IBM.
Language selection: English
Welcome
Release Information
Prerequisite Information
Installation Scenarios
Migration Information
Install IBM Tivoli Directory Integrator
Install IBM Tivoli Directory Integrator Password Synchronization Plug-ins
Exit

Welcome to the IBM Tivoli Directory Integrator 6.1.1 Launchpad

The IBM Tivoli Directory Integrator 6.1.1 Launchpad provides an overview of the TDI 6.1.1 release, install prerequisites, install scenarios and migration information. The Launchpad also provides links to detailed information about these and other related topics. Links to the IBM Tivoli Directory Integrator Installer and the IBM Tivoli Directory Integrator Password Synchronization Plug-ins Installer are also provided.

Web Sites:
- IBM Tivoli Directory Integrator Web site
- IBM Tivoli Directory Integrator 6.1.1 Documentation (Information Center)
- IBM Tivoli Directory Integrator support Web site
- IBM Tivoli Directory Integrator News Group

The choices on the left will choose TDI Launchpad panels:

Release Information
The Release Information panel contains a list of some of the new and improved features available this release, as well as links to documentation about the release.

Prerequisite Information
This panel contains links to information about platform support and hardware requirements.

Installation scenarios
This panel contains a description of the TDI components available for installation. You can install some or all of these components during installation. This panel also contains a description of the Password Synchronization Plugins components available for installation.

Migration Information
This panel contains a link to information about migrating from TDI 6.0 or 6.1 to 6.1.1. It also contains information about migrating the Cloudscape System Store.

Install IBM Tivoli Directory Integrator
This panel contains links to the IBM Tivoli Directory Integrator Installer, as well as links to installation, migration and supported platforms documentation. See for instructions on how to use the IBM Tivoli Directory Integrator Installer.

Install IBM Tivoli Directory Integrator Password Synchronization Plug-ins
This panel contains links to the IBM Tivoli Directory Integrator Password Synchronizer Plug-ins Installer, as well as links to installation and supported platforms documentation.

Note: This panel is not available on Linux PPC and Linux 390 platforms.

Exit Exits the Launchpad, without installing anything.

2. On the installation panel, click IBM Tivoli Directory Integrator **Installer**. This launches the installer. See "Using the platform-specific TDI installer" on page 10 for instructions on how to use the installer.

Launch the installer directly

You can launch the installer directly using the installation executable:

1. Locate the install executable file for your platform in the `tdi_installer` directory on the product CD (on i5/OS this directory is called `TDI_INST`):

Windows Intel
install_tdiv611_windows.exe

Windows AMD64/EM64T
install_tdiv611_amd64windows.exe

AIX install_tdiv611_aix.bin

Linux install_tdiv611_linux..bin

Linux AMD 64
install_tdiv611_amd64linux.bin

Power PC Linux
install_tdiv611_ppclinux.bin

z/OS Linux
install_tdiv611_zlinux.bin

Solaris
install_tdiv611_solaris.bin

HPUX PA-RISC
install_tdiv611_hpux.bin

HPUX Itanium
install_tdiv611_hpux_ia64.bin

i5/OS INST_TDI.SH

2. Double-click the executable, or type the executable name at the command prompt. This launches the installer. See "Using the platform-specific TDI installer" for information on how to use the installer.

Once you have launched the installer (using the Launchpad or by starting the platform-dependent installer directly), you are ready to begin the process of installing TDI.

Note: The default for installing the product, requires Administrator privilege.

Using the platform-specific TDI installer

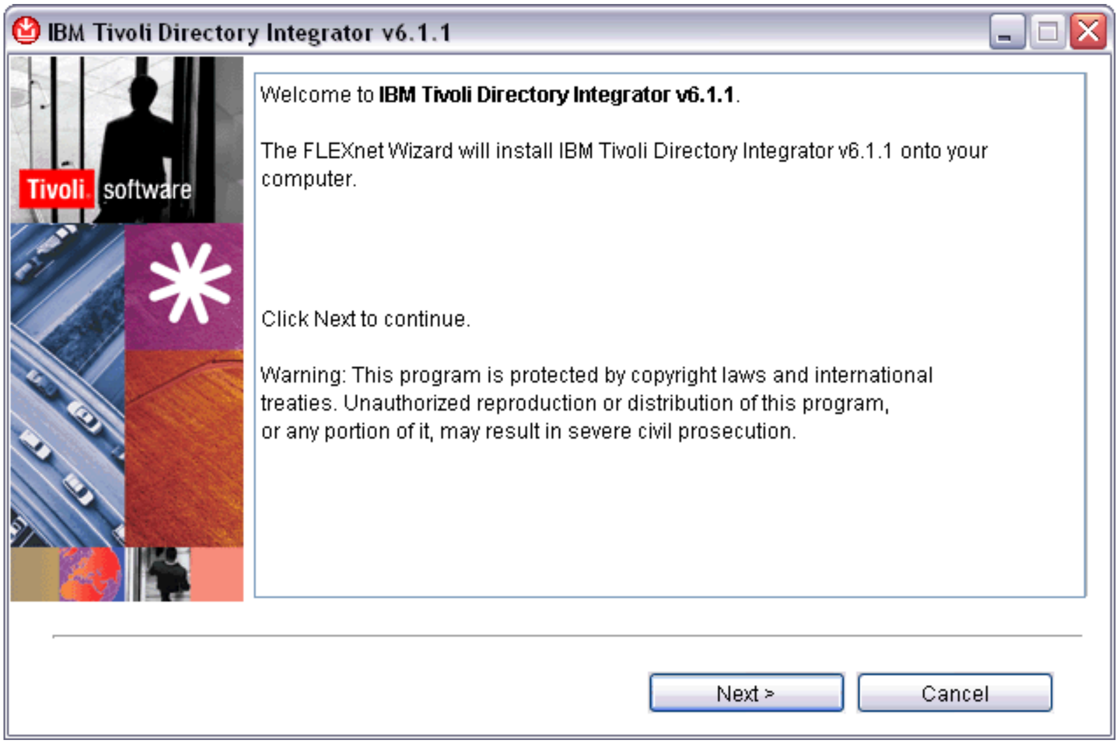
The platform-specific TDI installer is launched either form the Launchpad or from the command line. On i5/OS, this is the only way as there is no Launchpad.

The TDI installer can be used to install a new copy of TDI, add or remove a feature from an existing instance of TDI or upgrade a previous version of TDI. Each of these functions represent a different phase.

Installing a new version of TDI 6.1.1

TDI is either not present on the target system, or you chose to install a new version when prompted.

1. After starting the platform-specific installer, a number of progress bars stating "The FLEXnet Wizard is preparing the installation, please wait...." will appear.
2. After some time, you will see a panel stating "Initializing Installation Data...."
3. Eventually, the Welcome Panel will appear.


IBM Tivoli Directory Integrator v6.1.1

Welcome to **IBM Tivoli Directory Integrator v6.1.1**.

The FLEXnet Wizard will install IBM Tivoli Directory Integrator v6.1.1 onto your computer.

Click Next to continue.

Warning: This program is protected by copyright laws and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil prosecution.

The ″Welcome″ panel of the TDI Installer provides you with information about IBM Tivoli Directory Integrator 6.1.1. Click **Next** to continue.

4. After reading the Software license agreement, select **I accept the terms in the license agreement if you accept its terms**. Click **Next** to continue.
5. Select the installation type you prefer:
   - Select Typical to install the TDI Server, Config Editor, JavaDocs and Examples. This option does not give you the opportunity to install individual features.
   - Select Custom to choose individual features to install.
6. Click **Next**.
7. Select a directory where the product will be installed and click **Next**.

   Note: This panel is not displayed if you selected to upgrade by installing TDI 6.1.1 on top of TDI 6.0 or TDI 6.1.

   You cannot choose an install location of an existing installation of TDI 6.0, if you attempt this you will get an error: "Destination directory is the same as a previous version of Tivoli Directory Integrator. Please enter a new destination directory.″; and

   you cannot choose an install location of an existing installation (an "instance") of TDI 6.1, if you attempt this you will eventually see the error ″InstanceExists″.

   If you are installing a new copy of TDI 6.1.1 but you specify a location where an instance of TDI 6.1.1 already exists:

   a. If you do not select any new features, then you will get an error: ″Nothing selected to install, or the product is already installed at this location.″
   b. However, if you selected any new features to install, the install will proceed normally and the new features will be added to the existing instance.
8. If you are performing a custom installation, select the TDI components you want to install. See “Components in IBM Tivoli Directory Integrator” on page 4 for a list and descriptions of components available for installation. When you have finished making your selections, click **Next**.
9. If you are performing a typical installation, or if you selected to install the Runtime Server, you must select a Solutions directory. The Solutions directory is where the TDI Server and Config Editor locate your solutions, like Config files and properties files. Do the following:

   a.
      - **Use a TDI subdirectory under my home directory**. This creates a subdirectory under your home directory (in the rest of this documentation called the solution directory); for example for Windows: `C:\Documents and Settings\<username>\My Documents\TDI`
      - **Use Install Directory**. This is the directory you specified in step 7. On i5/OS, this choice is labelled **Use the TDI User Product Directory**.
      - **Select a directory to use**. Browse to or type the name of a directory. Any directory reachable on the local machine can be established as the default Solution Directory.
      - **Do not specify**. Use the current working directory at startup time.

   b. Click **Next**.
10. If you are performing a custom installation and you selected to install the Administration and Monitoring Console (AMC), but did not elect to install the embedded version of WebSphere, you must select a WebSphere Application server and profile:

    a. Select the WebSphere Application Server into which you want to load AMC.
       - You can select a currently installed WebSphere Application Server, if any are detected, by selecting the **Detected WebSphere Application Server** radio button and selecting the desired server from the drop-down menu.

Note: This installed version of WebSphere Application Server needs to be at level 6.0.2.13 or higher.

- You can create a custom location for the WebSphere Application Server by selecting **Custom location of WebSphere Application Server**. Use the Browse button to navigate to the desired location.
- To manually deploy AMC at a later time, select **Do not specify. I will manually deploy AMC at a later time**.

b. Click **Next**.

c. Select a profile in which to deploy AMC, from the drop-down menu. If you selected to install the AMC, select a WebSphere Application Server into which to deploy AMC.

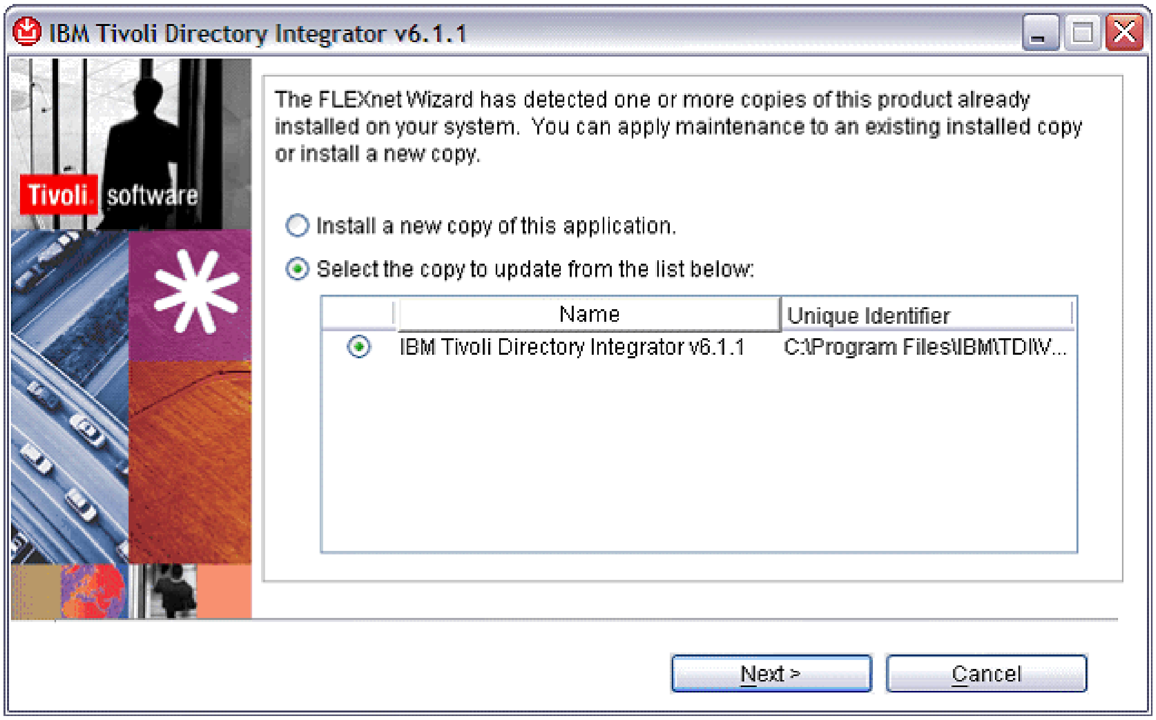
d. Click **Next**.

11. Review the installation information you have selected. Click **Back** to make any changes. When you are ready to begin installation, click **Install**.
12. When the product finishes installing, click **Next**.
13. Click **Finish** to complete the installation process.

Adding or removing features to an existing version of TDI 6.1.1

TDI 6.1.1 is present on the system, and the TDI 6.1.1 installer is invoked to add or remove features from the existing installation.

1. After starting the platform-specific installer, a number of progress bars stating ″The FLEXnet Wizard is preparing the installation, please wait....″ will appear.
2. After some time, you will see a panel stating ″Initializing Installation Data....″
3. The next panel will provide the option to install a copy of TDI 6.1.1 to a new location, or to update an existing copy of TDI 6.1.1. For this scenario, updating an existing copy will be chosen.



The FLEXnet Wizard has detected one or more copies of this product already installed on your system. You can apply maintenance to an existing installed copy or install a new copy.

○ Install a new copy of this application.

● Select the copy to update from the list below:

	Name	Unique Identifier
●	IBM Tivoli Directory Integrator v6.1.1	C:\Program Files\IBM\TDI\V...

4. You will be prompted to modify the existing instance of TDI 6.1.1 or remove it. Choose **Modify** and click **Next**.
5. You will be prompted to either add or remove features. Choose **Add Features** and click **Next**.

6. The Welcome Panel stating that you will be modifying a copy of TDI 6.1.1 will be displayed. From this point on, the steps are the same as described in “Installing a new version of TDI 6.1.1” on page 10 with these exceptions:
   a. The license panel will be skipped,
   b. The installation type panel will be skipped,
   c. The install location will be skipped.

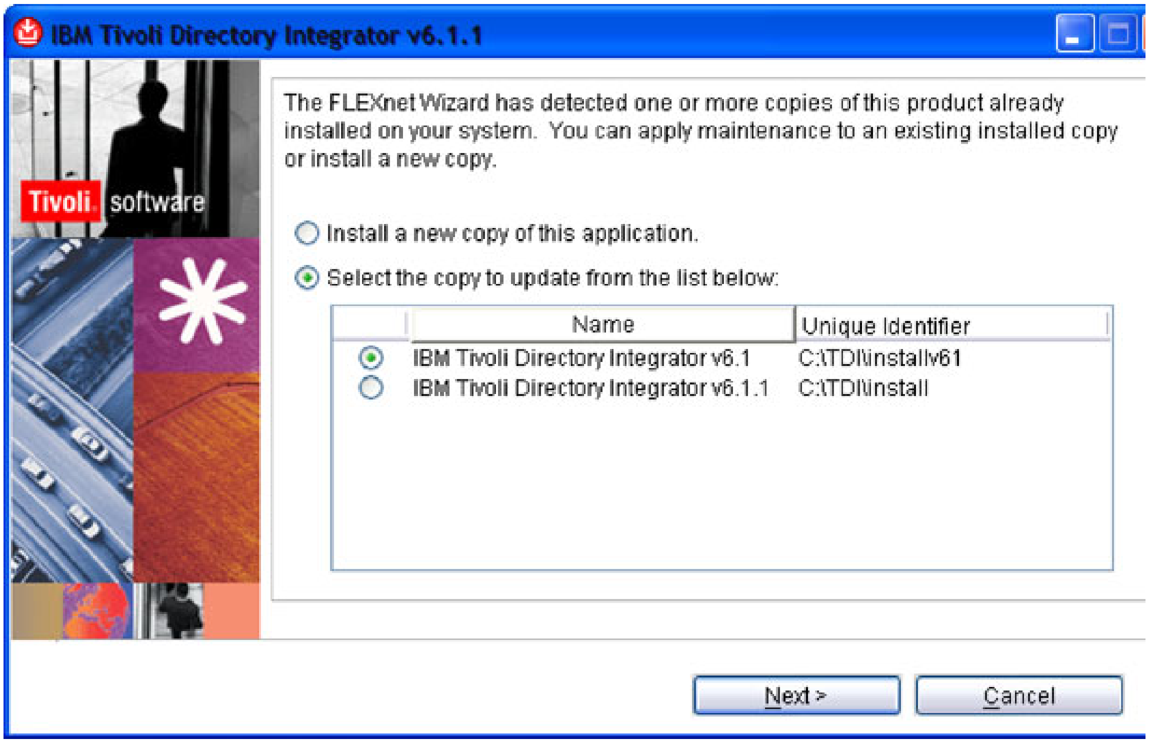
   You will see the feature selection panel, and the rest is the same.

Updating a previous version of TDI 6.1 to TDI 6.1.1

A previous version of TDI (either 6.0 or 6.1) is present on the system, and the TDI 6.1.1 installer is invoked to upgrade the existing installation.

1. After starting the platform-specific installer, a number of progress bars stating ″The FLEXnet Wizard is preparing the installation, please wait....″ will appear.
2. After some time, you will see a panel stating ″Initializing Installation Data....″
3. The next panel will provide the option to install a copy of TDI to a new location, or to update an existing copy of TDI 6.1 For this scenario, updating an existing copy will be chosen.

   Note: If a copy of TDI 6.1.1 is present along with TDI 6.1, both instances will show up in this panel. Choosing the TDI 6.1.1 instance would be the same as adding/removing a feature.



The FLEXnet Wizard has detected one or more copies of this product already installed on your system. You can apply maintenance to an existing installed copy or install a new copy.

○ Install a new copy of this application.

● Select the copy to update from the list below:

	Name	Unique Identifier
●	IBM Tivoli Directory Integrator v6.1	C:\TDI\installv61
○	IBM Tivoli Directory Integrator v6.1.1	C:\TDI\install

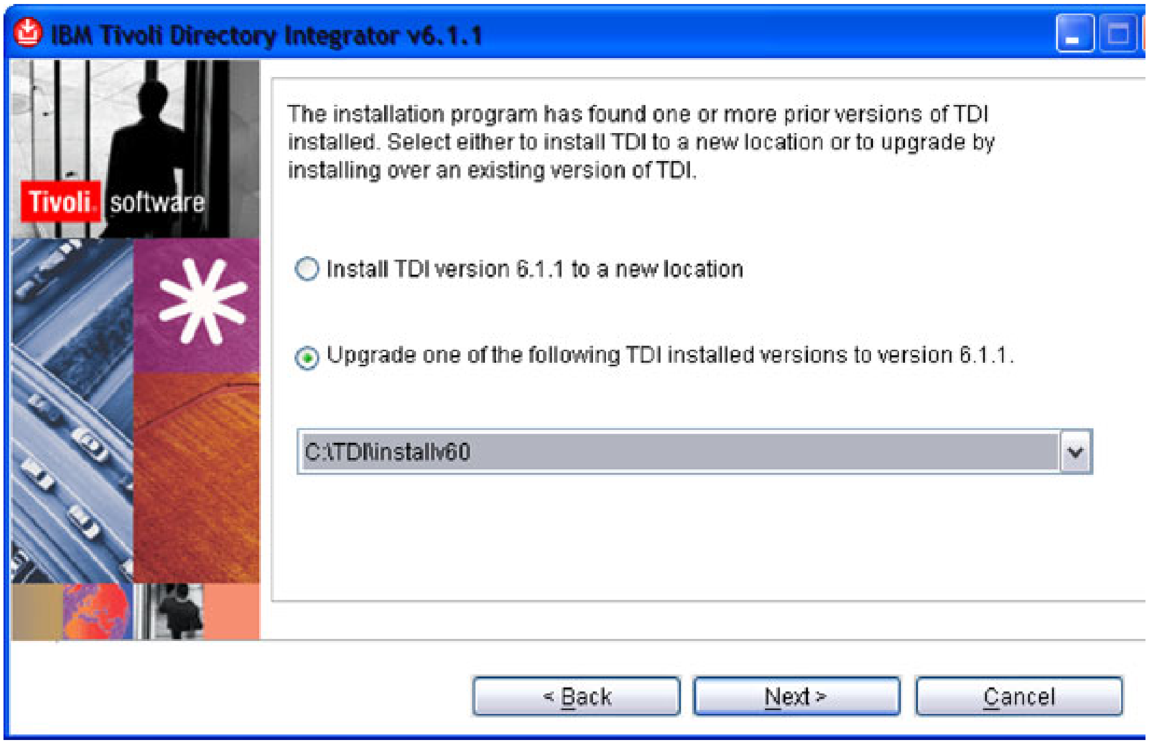
4. The Welcome Panel stating that you will be updating a copy of TDI 6.1 will be displayed. From this point on, the steps are the same as described in “Installing a new version of TDI 6.1.1” on page 10 with these exceptions:
   a. The license panel will be skipped,
   b. The installation type panel will be skipped,
   c. The install location will be skipped.

   You will see the feature selection panel, and the rest is the same.

Updating a previous version of TDI 6.0 to TDI 6.1.1

In order to update a previous version of TDI 6.0 to TDI 6.1.1, you must follow the "Installing a new version" steps.

1. Choose to install to a new location.
2. After the Welcome and the System Checks you will be presented with the following panel:



The installation program has found one or more prior versions of TDI installed. Select either to install TDI to a new location or to upgrade by installing over an existing version of TDI.

○ Install TDI version 6.1.1 to a new location

● Upgrade one of the following TDI installed versions to version 6.1.1.

C:\TDI\installv60

   Choose "Upgrade one of the following TDI installed versions to version 6.1.1" and select the TDI installation from the pull down menu. Click **Next**.

   Note: There is no version detection and automated upgrade for versions prior to TDI 6.0.
3. You will see the install type selection panel and if you choose Custom, the feature selection panel. You will not be prompted for an install location, or TDI Solutions directory information.
4. The rest of the steps are the same as "Installing a new version" until just before step 11 on page 12. If the TD 6.0 install used a Cloudscape database as a System Store in the context of TDI 6.0, an information panel appears with the following message:

   "The Cloudscape system store data will be migrated to Cloudscape v10.1.3.1 to make it compliant with IBM Tivoli Directory Integrator 6.1.1".

   No action is required. Click **Next**.
5. The rest of the steps are the same as "Installing a new version".

Installing using the command line

The following command line options are supported by the TDI 6.1.1 installer:

-console:

Specifies to use the console interface mode, where messages during installation are displayed on the Java console and the wizard is run in console mode.

```
<Install Wizard>/install_tdiv611_windows.exe –console
```

-options-record

Specifies that the TDI Install Wizard should automatically generate a response file for the project after the completion of the installation/uninstallation.

```
<Install Wizard>/install_tdiv611_windows.exe –options-record <Response File Name>
```

-options

Specifies that a response file be used to execute the installation/uninstallation of TDI. A response file is usually used when a silent installation is run (see the next option).

```
<Install Wizard>/install_tdiv611_windows.exe –options <Response File Name>
```

Note: The directory *<TDI_install_directory>*/examples/installer contains a number of example response files for various installation and de-installation scenarios.

-silent Specifies to install or uninstall the product in silent mode, where the installation/uninstallation is performed with no user interaction. The **–options** command line option is used here to specify what response file to use.

```
<Install Wizard>/install_tdiv611_windows.exe –silent –options <Response File Name>
<Install Location>/_uninst/uninstaller.exe -silent -options <Uninstall Response>
```

-is:javahome <*java home directory***>**

Specifically tells the launcher the home directory location of the JVM to use. The JVM that is specified must be at the 1.4.2 level (currently we are bundling 1.5 SR3).

```
 <Install Wizard>/install_tdiv611_windows.exe –is:javahome c:\Java142
```

-is:javatimer *seconds*

On certain platforms, especially EMC VMware, VPars and NPars, if the bundled JVM fails to verify, you may need to add the command line option -is:javatimer <seconds> to extend the JVM verification window.

-is:log <*filename***>**

This option is useful for setup launchers that hide the Java console because it logs the detailed information about the launcher's processing, including the actual Java commands that were used to start the Java program, to the specified fileName. This includes all of the ″std out″ and ″std err″ messages from the Java process.

```
<Install Wizard>/install_tdiv611_windows.exe –is:log c:\temp\Log.txt
```

-is:tempdir <*directory***>**

Sets the path to the temporary directory to which the launcher should write its temporary files. If the specified directory does not exist or is not a directory, the launcher will use the system temp directory instead, and no error message is provided.

```
<Install Wizard>/install_tdiv611_windows.exe –is:tempdir c:\privateTemp
```

The following command line options are unique to the TDI Install Wizard:

-siinstall <*true***>**

Specifies whether you are required to have administrative (root) authority. When set to "true" administrative authority is not required, and non-administrative users will be allowed to install TDI; however, the directory in which TDI is installed must be writable for the person performing the install. For example:

```
 <Install Wizard>/install_tdiv611_windows.exe –siinstall true
```

Note: Your must have the proper authority to write to the install directory specified in the Installer; and the use of this mode requires that you use the -console option. Also, on AIX and i5/OS, administrative (root) authority is always required.

-V TDI_BackupAMC

This parameter should only be passed in on an uninstall. This parameter is provided for future migration considerations.

```
        <Install Location>/_uninst/uninstaller.exe –V TDI_BackupAMC=true
```

Performing a silent install

To perform a silent installation you must first generate a response file. To generate this file, perform a non-silent install with the `-options-record` option specified; for example:

```
<Install Wizard>/install_tdiv611_windows.exe –options-record <Response File Name>
```

The response file is created in the directory that you specify during installation.

Note: The directory <*TDI_install_directory*>/examples/installer contains a number of example response files for various installation and de-installation scenarios.

Once the response file is created, you can install silently using the following command:

```
<Install Wizard>/install_tdiv611_windows.exe –silent –options <Response File Name>
```

Note: The examples use the Windows platform install executable. See "Launching the appropriate installer" on page 7 for a list of executable names for each supported platform.

Note: On Windows, a shortcut to the Config Editor is created on the desktop.

Installing local Help files

As the IBM Tivoli Directory Integrator installer does not contain any user documentation, other than the Javadocs API documentation (which can be displayed by selecting the **Help>Low Level API** menu in the Config Editor). IBM provides the user documentation in online form in an Infocenter, at http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/welcome.htm

IBM Tivoli Directory Integrator is equipped with code[1] to provide you with context-dependent online Help that you can launch from the Config Editor (CE). By default, this code will resolve the documentation from the online Infocenter as referenced above. You can, however, install the documentation locally, such that you are not dependent upon the Internet to be able to read it.

These are the steps you must take to install documentation locally:

- All the manuals are bundled together in one zipped directory, which when unzipped will contain an Eclipse *Document plugin*.
- Download the manuals, in their zipped form, from the IBM TDI documentation site, at http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/welcome.htm
- Un-zip the documentation package, *di_plugins-6.1.1.zip* into the right place: *installation_directory/ibm_help/eclipse/plugins* folder (or unzip somewhere else, and move into the right place). The package contains the actual TDI documentation in `com.ibm.IBMDI.doc_6.1.1`, alongside a number of other directories whose names end in .doc – all those directories should be at the same aforementioned *plugins* level.
- The location of the documentation that the CE tries to access is set in the `global.properties` file , which resides at the root level of the installation directory of IBM Tivoli Directory Integrator, or `solutions.properties` in the Solution Directory. By default, this points to the Online Infocenter, but if you comment the line

  ```
  ## Name of help server, comment out if you want local help system
  com.ibm.di.helpHost=publib.boulder.ibm.com/infocenter/tiv2help/index.jsp?topic=
  ```

  such that it reads

  ```
  ## Name of help server, comment out if you want local help system
  #com.ibm.di.helpHost=publib.boulder.ibm.com/infocenter/tiv2help/index.jsp?topic=
  ```

[1] The help system is powered by Eclipse technology. (http://www.eclipse.org)

then next time you run the CE and launch Help, it will start a local task to serve the documentation, from the content in the *plugins* directory.

- The location of the documentation server that AMC tries to access is set in AMC's web.xml file. Open the web.xml file which will be located in the WEB-INF folder of tdiamc webapp and mention the help server's IP address (or hostname) and port for both occurrences of the following attributes: InfocenterHostName and InfocenterPort.

After you install the documentation in the *plugins* directory as outlined above, you can also decide to host the documentation on that machine for other installations of IBM Tivoli Directory Integrator in your neighborhood. In the *installation_directory/ibm_help* directory there are a number of .bat files (Windows) or .sh files (Unix/Linux) that enable you to do this.

IC_start.bat or IC_start.sh

If you run this script, this will start an Infocenter on http://your_IP_address:8888

By editing this file, you can change the port number from the default, 8888; if you want to change this, to for example 80, change "-port 8888" to "-port 80" instead of -port 8888. On those clients that are trying to access this Infocenter, the port must match another property in the `global.properties` or `solution.properties` file, `com.ibm.di.helpPort` – its default is set to 80. Also, the `com.ibm.di.helpHost` property should read something like `infocenter_IP_address/help`, where infocenter_IP_address is the address of your local Infocenter.In addition, in order for AMC to find this Infocenter, you need to update the parameters InfoCenterHostname and InfoCenterPort attributes in AMC's configuration file, `web.xml`, to match the values above.

IC_stop.bat or IC_stop.sh

Stops the help system, a Java program, that serves the local Infocenter.

help_start.bat or help_start.sh

Similar to IC_start, except the port used will be a random one, and it will also launch a local browser showing the start page. As the port is random, unsuitable for use other than on the local machine.

help_stop.bat or help_stop.sh

Stop the local Java task that was started by WebSphere_help_start.

Uninstalling

You can uninstall TDI in its entirety, or uninstall only certain components.

Launching the uninstaller

To uninstall IBM Tivoli Directory Integrator, you must first launch the uninstaller:

1. Navigate to the TDI _uninst directory, for example:

   `<install_path>/_uninst`

2. Launch the uninstaller by executing the uninstall executable.

   For Windows platforms, the uninstall executable is called uninstaller.exe. For all other platforms, the uninstall executable is called uninstaller.bin, except for i5/OS where it is called uninstaller.sh.

   Note: On Windows platforms, you can also uninstall using the Add/Remove programs Control Panel.

Uninstalling individual TDI components

To uninstall certain TDI components:

1. On the "Welcome" panel, select the **Remove Features** radio button
2. Click **Next**.
3. Installed features are indicated by check marks. To uninstall a feature, click the check mark to deselect it. The check mark will be removed.

4. Click **Next**.
5. Verify the list of features to be removed under **Features to Remove**.
6. Click **Back** to make any changes. When you are ready to uninstall, Click **Uninstall**.
7. When the uninstall completes, click **Finish**.

Uninstalling IBM Tivoli Directory Integrator

To uninstall all IBM Tivoli Directory Integrator components:

1. On the "Welcome" panel, select the **Remove** radio button
2. Click **Next**.
3. Verify that you want to uninstall IBM Tivoli Directory Integrator.
4. Click **Back** to make any changes. When you are ready to uninstall, click **Uninstall**.
5. When the uninstall completes, click **Finish**.

Performing a silent uninstall

To perform a silent uninstall of IBM Tivoli Directory Integrator you must first generate a response file. To generate this file, you must perform a full GUI uninstall with the `-options-record` option specified; for example:

```
<install_path>/_uninst/uninstaller.exe –options-record  <UninstallResponseFileName>
```

The response file is created in the directory that you specify during uninstallation.

Note: The directory <*TDI_install_directory*>/examples/installer contains a number of example response files for various installation and de-installation scenarios.

Once the response file is created, you can uninstall silently using the following command:

```
<install_path>/_uninst/uninstaller.exe -silent -options  <UninstallResponseFileName>
```

Note: The examples use the Windows platform uninstall executable.

Default install locations

IBM Tivoli Directory Integrator installs to the following default locations:

Windows platforms
c:\Program Files\IBM\TDI\V6.1.1

Linux and UNIX platforms (AIX, HPUX, Solaris)
/opt/IBM/TDI/V6.1.1

i5/OS /QIBM/ProdData/TDI/V6.1.1

Distribution components

Here is the list of components (.jar and .dll files) installed on your computer with IBM Tivoli Directory Integrator.

.jar files are found in the jars/ subdirectory. Connectors, EventHandlers, Function Components and Parsers are found in separate subdirectories and not described here.

Root directory

The Root directory is the directory where TDI 6.1.1 is installed, see also “Default install locations.”

IDILoader.jar
TDI classloader.

ibmditk
Executable file for graphical user interface (GUI), also known as Config Editor (CE). Non-Windows platforms only.

Note: The PATH system variable is not set during installation. Set the path so commands such as ibmditk can work in a directory other than the IBM Tivoli Directory Integrator root directory.

ibmditk.bat
Like **ibmditk** above, except for Windows platforms.

ibmdisrv
Executable file for the TDI server (non-Windows platforms only).

Note: The PATH system variable is not set during installation. Set the path so commands such as ibmdisrv can work in a directory other than the IBM Tivoli Directory Integrator root directory.

ibmdisrv.bat
Like **ibmdisrv** above, except for Windows platforms.

ibmdicwd.bat
Necessary on Windows to change the working directory to the value of the first parameter.

testserver.der
Exported sample server certificate, ready to be imported in a truststore.

testserver.jks
Sample server keystore and truststore. This file is referenced as an example in `global.properties`.

isInfo.is
File installed by Installer to provide support for persistant variables.

idisrv.sth
The stash file that contains the Server keystore password values encrypted with AES128 with a fixed key.

Root directory/amc

This folder contains only one file, `tdiamc.war`. This folder is created only if the AMC option is selected during the TDI Installation process. This WAR file is placed so that administrators can choose to deploy this file on any J2EE compliant Web container like IBM WebSphere Application Server, Tomcat, etc.

Root directory/bin

This folder contains various utility scripts for TDI. The list of various files is:

collect.bat(sh)
This script collects serviceability info.

createstash.bat(sh)
A command line utility for creating a stash file

dirlist.txt
List of directories that will be backed up by the collect script

filelist.txt
List of files that will be backed up by the collect script

logcmd.bat(sh)
Utility to configure JLog trace properties dynamically

tdisrvctl.bat(sh)
Remote command line interface to manage TDI Configs and AssemblyLines.

tdimiggbl.bat(sh)
Command to migrate the global.properties file to a TDI 6.1.1 compatible version. Use the -? option for printing the usage of the command.

Root directory/bin/amc

This folder contains scripts for AMC and Action Manager. This folder will be created only if the AMC option is selected during the TDI 6.1.1 Installation process.

backupamc.bat(sh)
Backup script for storing AMC data and configuration files.

createProfile.bat(sh)
Wrapper script over wasprofile.bat(sh) for creating a WAS profile.

createWasExpProfile.bat
This script creates the profile in WAS.

deleteProfile.bat(sh)
Wrapper script over wasprofile.bat(sh) for deleting a WAS profile.

install.bat(sh)
Wrapper script over wsadmin command for installing AMC into amcprofile. Modify the APPSRV_INSTALLROOT variable to point to custom WAS install location and AMC_PROFILE variable to point to custom profile if you wish to install AMC on a different WAS server instance and profile.

migrateamc.bat(sh)
Wrapper script which internally calls backupamc, uninstall, stopServer, install, restoreamc and startServer to perform a migration of amc. You can instead also choose to run each of these commands individually for better control and debugging.

reinstall.bat(sh)
Wrapper script which internally calls uninstall, stopServer, install, startServr to perform a re-installation of AMC. Users can instead also choose to run each of these commands individually for better control and debugging.

restoreamc.bat(sh)
Script for restoring contents of AMC backed up by the backupamc script.

start_tdiamc.bat(sh)
Script for starting AMC on WAS. This is a wrapper script which calls startServer. Modify the APPSRV_INSTALLROOT variable to point to custom WAS install location and AMC_PROFILE variable to point to custom profile if you wish to start AMC on a different WAS server instance and profile.

startAM.bat(sh)
Script for starting Action Manager.

startNetworkServer.bat(sh)
Script for starting Cloudscape Network Server. Both Action Manager and AMC talk to a Cloudscape database in network mode. For remote Action Manager to connect to the Cloudscape database, it needs to be running in Network mode. For this reason, when administrators wish to start the Cloudscape network server and want remote Action Managers to connect to AMC and Cloudscape database on this machine, they should use this utility.

Note: This utility starts a Cloudscape network server running on localhost and port 1527. If you have modified AMC and Action Manager to use a Cloudscape Network Server on a different port, then modify this script before running the script. Modify **hostname** ″localhost″ to IP address of the machine in the script for other machines in the network to identify this machine.

stop_tdiamc.bat(sh)
Script for stopping AMC on WAS. This is a wrapper script which calls stopServer. Modify the APPSRV_INSTALLROOT variable to point to custom WAS install location and AMC_PROFILE variable to point to custom profile if you wish to stop AMC on a different WAS server instance and profile.

stopNetworkServer.bat(sh)
Script for stopping Cloudscape Network Server. Both Action Manager and AMC talk to a Cloudscape database in network mode. When Action Manager is started, it attempts to connect to the AMC DB in network mode. If the AMC DB is not started, then it starts the network server. But, on termination of Action Manager, the network server is not stopped – since there AMC may be already connected to Cloudscape. For this reason, when administrators wish to stop the Cloudscape network server, they should use this utility.

Note: This utility stops any Cloudscape network server running on localhost and port 1527. If you have modified AMC and Action Manager to use a Cloudscape Network Server on a different port, then modify this script before running the script.

tdimigam.bat(sh)
Command to migrate the am_config.properties file of TDI 6.1 Action Manager, to one compatible with TDI 6.1.1 Action Manager. Use the -? option for printing the usage of the command.

uninstall.bat(sh)
Wrapper script over the wsadmin script for uninstalling AMC from WAS. Note: It is mandatory for the WAS server to be running for being able to uninstall any Web Application.

Root directory/bin/amc/Action Manager: This folder contains all configuration files, log files and jars for running Action Manager.

am_config.properties
Action Manager configuration file.

am_logging.properties
The Action Manager logging configuration file.

tdimigam-log4j.properties
The log4j configuration file that controls the logging properties of the tdimigam.bat(sh) command.

testadmin.jks
The SSL Truststore and Keystore file referenced by Action Manager.

logs/ folder
This folder will contain the logs for every run of Action Manager.

jars/ folder
This folder contains the jar files necessary for Action Manager to run. List of jar files in this folder are:

ITLMToolkit.jar
action_manager.jar
db2jcc.jar
db2jcc_license_c.jar
derby.jar
derbyLocale_de_DE.jar
derbyLocale_es.jar
derbyLocale_fr.jar
derbyLocale_it.jar
derbyLocale_ja_JP.jar
derbyLocale_ko_KR.jar
derbyLocale_pt_BR.jar
derbyLocale_zh_CN.jar

derbyLocale_zh_TW.jar
derbyclient.jar
derbynet.jar
derbytools.jar
diserverapi.jar
diserverapirmi.jar
icu4j_3_6.jar
log4j-1.2.8.jar
miconfig.jar
migam.jar
miserver.jar
mmconfig.jar
tdiresource.jar

Root directory/AppServer

This folder is created by the TDI 6.1.1 installer only if the option of installing Embedded WAS Express 6.0.2 is chosen during installation. This folder contains the Embedded WAS Express image, and can be used to run AMC. The TDI 6.1.1 installer can automatically deploy AMC on this image. AMC is installed on the following location in such a case: `AppServer/profiles/amcprofile/installedApps/DefaultNode/tdiamc.war.ear/tdiamc.war/`

Root directory/doc

This folder is created by the TDI 6.1.1 installer only if the option of installing Java Docs is chosen. This folder contains the low level Java documentation of the various classes and methods in TDI 6.1.1. For viewing the Java documentation point your browser to the `docs/api/index.html` file (or select **Help->Low Level API** from the CE.)

Root directory/etc

This folder contains all the configuration files for the TDI 6.1.1 Server and its components like the Config Editor and CLI.

build.properties
Contains the TDI Build information, build date, version, etc.

ce-log4j.properties
A log4j properties file for controlling the logs generated by the TDI 6.1.1 Config Editor (ibmditk).

CSServersInfo.xml
An XML repository file for storing multiple System Store configuration details.

derby.properties
The default configuration file for Cloudscape System Store shipped with TDI 6.1.1. For more details on this file refer the Cloudscape v10 documentation on the IBM Website .

executetask.properties
This file controls the logging strategy of the TDI server (ibmdisrv) when started from Configuration Editor (ibmditk). This is a log4j configuration file.

global.properties
This file is the main configuration file for TDI 6.1.1.

global.properties.v611
This is a placeholder for a sample file of TDI 6.1.1. This file is non-empty only if you have chosen to upgrade from previous versions of TDI to TDI 6.1.1. In such cases, since the `global.properties` file is a migrated file, it would be helpful for you to compare your migrated file to `global.properties.v611` to see how it is different from doing a clean install. This file can be a helpful reference.

ibmdi.ico
The icon file for TDI 6.1.1.

ITDIJ060100.sys
This is product signature (license) file used by the ITLM agent to recognize TDI.

jlog.properties
The JLog configuration file for TDI 6.1.1

log4j.properties
The Log4j configuration file for controlling theTDI 6.1.1 Server (ibmdisrv) logging.

reconnect.rules
A text file where you can define reconnect rules on how reconnect exceptions should be handled by TDI.

tdisrvctl-log4j.properties
The Log4j configuration file for controlling the log traces generated by the tdisrvctl command line utility.

act-jlog.properties
Describes the properties file format used by the ACT logging component.

tdimiggbl-log4j.properties
Controls the logging strategy for the `tdimiggbl` command line utility.

Root directory/classes

This folder is intended to contain user-provided class files that are loaded by the system class loader. This folder can be used for specifying custom classes which must be loaded by the system class loader.

For example, in order to use SSL with Domino and the Domino Connectors (IIOP Session) one needs to put a Domino-generated class in this folder.

In order for a class file to be loaded by the system class loader, the class file needs to be copied to this folder. If the class is inside a Java package, then the class file must be put in the corresponding folder under the "classes" folder. For example, if a class file is contained in a Java package named "com.ibm.di.classes", then the class file must be put inside the "<TDI_Install_Folder>/classes/com/ibm/di/classes" folder.

Only the class files in the "classes" folder are loaded. That means that if a jar file is located in this folder, it will not be loaded at all. If the classes are packed in a jar file, then these classes need to be extracted from the jar file into the "classes" folder. After TDI installation this folder is empty.

Root directory/examples

This folder contains TDI examples. Each example is in a separate folder. Every example contains a "readme.txt" file with additional details about how to setup and run the example.

Root directory/installLogs

This folder will contain the AMC installation logs if AMC is chosen to be installed during TDI 6.1.1 installation.

Root directory/libs

On the Windows platform, this folder contains DLL files for a few of the TDI 6.1.1 components. The list of installed files is: COMProxy.dll, domchdet.dll, NTEventLogAppender.dll, WindowsUsers.dll.

Root directory/logs

The logs generated by TDI Server, Config Editor and Command Line Interface(tdisrvctl) go into this folder. The Server logs by default are ibmdi.log, Config Editor logs are ibmditk.log, and tdisrvctl logs are tdisrvctl.log.

Root directory/performance

Utilities for Unix platforms. Not installed on Windows.

benchmark.properties
Configuration file for specifying benchmark and throughput properties.

ibmdibenchmark.sh
Runs the benchmark utility.

ibmdisrvtp.sh
Runs the throughput utility.

tdiperfhead.sh
Common script used by ibmdibenchmark.sh and ibmdisrvtp.sh.

Root directory/tools/CSMigration

Contains Cloudscape v5 to Cloudscape v10 migration utility. You can use this utility to automatically migrate your Cloudscape system store of TDI 6.0 to TDI 6.1.1. This folder contains two files:

migrateCS.bat(sh)
The script to run for migration

migratetoderby.jar
Library jar file used by the migrateCS script.

Root directory/XSLT/ConfigReports

Contain XSL files, images and translation folder. These are used to generate the various Config Reports for an AssemblyLine. The Config Report option is shown to the user when he right clicks on an AssemblyLine in the pop-up context menu.

Root directory/_uninst

This directory contains the uninstaller executable for the application and other related files the uninstaller executable depends on.

jvm/ folder – This is the JRE 1.4.2 SR4 folder used by the installer tools. It contains a doc folder, a jre folder and a COPYRIGHT file. This jvm folder is unrelated to the “Root directory/jvm” folder of the actual TDI 6.1.1 installation.

Root directory/jvm

This is the IBM JRE 1.5.0 SR3 folder used by TDI 6.1.1. It contains a doc folder, a jre folder and a COPYRIGHT file.

Root directory/license

This folder contains TDI 6.1.1 License files and notices.

Root directory/ibm_help

This folder contains the IBM Eclipse Help System. This gets installed only if the IBM Eclipse Help System (IEHS) option is chosen during TDI Installation. This help system can be used to host TDI 6.1.1 documentation locally. See also “Installing local Help files” on page 16.

IC_start.bat(sh)
Script to start the Info Center. This InfoCenter will start an Embedded Application/WebServer and listen for Browser requests (HTTP) on the port specified by the –p option of the script.

IC_end.bat(sh)
Script to stop the Info Center.

help_start.bat(sh)
Script to start the Help System locally on a randomly generated port.

help_end.bat(sh)
Script to stop the local Help System.

eclipse/
Contains the Eclipse binaries for running the Help System. The TDI Documentation plugin must be placed in the eclipse/plugin folder for it to show up in the IBM InfoCenter Welcome Page.

Root directory/xsl

XSL files that are part of the IBM Tivoli Directory Integrator 6.1.1 Component Suite for SAP R/3.

Root directory/serverapi

This folder contains several files used by the Server API.

- **cryptoutils.bat –** Command line utility used for encrypting/decrypting TDI configurations and user registry.
- **registry.enc** - Encrypted version of the "registry.txt".
- **registry.txt** - Example for a Server API user registry file.
- **testadmin.der –** Exported certificate from "testadmin.jks".
- **testadmin.jks** - Example keystore/truststore file for a Server API remote client.

Root directory/win32_service

This folder contains files needed to run TDI as a Windows service.

- **ibmdiservice.exe –** Windows service implementation.
- **ibmdiservice.props –** Windows service configuration file.
- **log4j.properties –** Defines logging strategy for the Windows service.

Root directory/jars

Table 1.

Location	Filename	Description
jars\3rdparty\IBM	auibase.jar	AUIML Library
jars\3rdparty\IBM	com.ibm.mqjms.jar	WebSphere MQ classes for Java Message Service
jars\3rdparty\IBM	db2jcc.jar	IBM Cloudscape
jars\3rdparty\IBM	db2jcc_license_c.jar	IBM Cloudscape
jars\3rdparty\IBM	derby.jar	IBM Cloudscape
jars\3rdparty\IBM	derbyclient.jar	IBM Cloudscape
jars\3rdparty\IBM	derbyLocale_de_DE.jar	IBM Cloudscape
jars\3rdparty\IBM	derbyLocale_es.jar	IBM Cloudscape
jars\3rdparty\IBM	derbyLocale_fr.jar	IBM Cloudscape
jars\3rdparty\IBM	derbyLocale_it.jar	IBM Cloudscape

Table 1. (continued)

Location	Filename	Description
jars\3rdparty\IBM	derbyLocale_ja_JP.jar	IBM Cloudscape
jars\3rdparty\IBM	derbyLocale_ko_KR.jar	IBM Cloudscape
jars\3rdparty\IBM	derbyLocale_pt_BR.jar	IBM Cloudscape
jars\3rdparty\IBM	derbyLocale_zh_CN.jar	IBM Cloudscape
jars\3rdparty\IBM	derbyLocale_zh_TW.jar	IBM Cloudscape
jars\3rdparty\IBM	derbynet.jar	IBM Cloudscape
jars\3rdparty\IBM	derbytools.jar	IBM Cloudscape
jars\3rdparty\IBM	dsml.jar	ITDS DSML Library
jars\3rdparty\IBM	dsml2.jar	DSML Library
jars\3rdparty\IBM	enroleagent.jar	ITIM DAML Library
jars\3rdparty\IBM	help.jar	IBM Eclipse-based Help System (IEHS)
jars\3rdparty\IBM	ibmjms.jar	JMS interfaces
jars\3rdparty\IBM	ibmjs.jar	IBM Script Engine
jars\3rdparty\IBM	IBMLDAPJavaBer.jar	IBMLDAPJavaBer
jars\3rdparty\IBM	icu4j_3_6.jar	ICU4J
jars\3rdparty\IBM	ITLMToolkit.jar	IBM Tivoli License Manager toolkit
jars\3rdparty\IBM	jffdc.jar	JFFDC
jars\3rdparty\IBM	jlog.jar	JLOG
jars\3rdparty\IBM	MQeBase.jar	MQ Everyplace
jars\3rdparty\IBM	MQeJMS.jar	MQ Everyplace
jars\3rdparty\IBM	MQeSecurity.jar	MQ Everyplace
jars\3rdparty\IBM	regex4j.jar	Regular Expression Library
jars\3rdparty\IBM	remoteaccess.jar	RXA (IBM Tivoli Remote Execution and Access)
jars\3rdparty\IBM	remoteaccess-as400.jar	RXA (IBM Tivoli Remote Execution and Access)
jars\3rdparty\IBM	snmp.jar	SNMP Library
jars\3rdparty\IBM	ssh.jar	RXA (IBM Tivoli Remote Execution and Access)
jars\3rdparty\IBM	viewer.jar	Logxml Libraries
jars\3rdparty\IBM	wsdl4j-1.5.1.jar	Apache Axis
jars\3rdparty\others	activation.jar	JavaBeans Activation Framework
jars\3rdparty\others	ACTCBEWrapper.jar	ACT Engine
jars\3rdparty\others	ACTCommon.jar	ACT Engine
jars\3rdparty\others	ACTParser.jar	ACT Engine
jars\3rdparty\others	ACTEngine.jar	ACT Engine
jars\3rdparty\others	ACTMsg.jar	ACT Engine
jars\3rdparty\others	ACTTECWrapper.jar	ACT Engine
jars\3rdparty\others	antlr.jar	ANTLR Parser Generator
jars\3rdparty\others	axis.jar	Apache Axis
jars\3rdparty\others	axis-ant.jar	Apache Axis

Table 1. (continued)

Location	Filename	Description
jars\3rdparty\others	axis-schema.jar	Apache Axis
jars\3rdparty\others	commons-discovery-0.2.jar	Apache Axis
jars\3rdparty\others	commons-logging-1.0.4.jar	Apache Axis
jars\3rdparty\others	hl14.jar	TPTP Platform
jars\3rdparty\others	hlcbe101.jar	TPTP Platform
jars\3rdparty\others	hlcore.jar	TPTP Platform
jars\3rdparty\others	jaxrpc.jar	Java API for XML-Based RPC (JAX-RPC)
jars\3rdparty\others	jakarta-oro-2.0.8.jar	Used by Castor FCs
jars\3rdparty\others	jaxrpc.jar	Used by Castor FCs
jars\3rdparty\others	jt400.jar	Access library for DB2 on i5/OS
jars\3rdparty\others	tptpcore.jar	RAC Connector support library
jars\3rdparty\others	jlanclient.jar	RXA (IBM Tivoli Remote Execution and Access)
jars\3rdparty\others	log4j-1.2.8.jar	LOG4J
jars\3rdparty\others	mail.jar	Sun library for POP/IMAP
jars\3rdparty\others	saaj.jar	SOAP with Attachments API for Java (SAAJ)
jars\3rdparty\others\emf	org.eclipse.emf.common_2.2.1.v200609210005.jar	Eclipse Modeling Framework (EMF)
jars\3rdparty\others\emf	org.eclipse.emf.commonj.sdo_2.1.0.v200609210005.jar	Eclipse Modeling Framework (EMF)
jars\3rdparty\others\emf	org.eclipse.emf.ecore.change_2.1.0.v200609210005.jar	Eclipse Modeling Framework (EMF)
jars\3rdparty\others\emf	org.eclipse.emf.ecore.sdo_2.2.0.v200609210005.jar	Eclipse Modeling Framework (EMF)
jars\3rdparty\others\emf	org.eclipse.emf.ecore.xmi_2.2.1.v200609210005.jar	Eclipse Modeling Framework (EMF)
jars\3rdparty\others\emf	org.eclipse.emf.ecore_2.2.1.v200609210005.jar	Eclipse Modeling Framework (EMF)
jars\3rdparty\others\emf	org.eclipse.xsd_2.2.1.v200609210005.jar	Eclipse Modeling Framework (EMF)
jars\ce	miadmin.jar	TDI Config Editor
jars\common	cli.jar	Command Line Utility
jars\common	diserverapi.jar	Server API
jars\common	diserverapirmi.jar	Server API
jars\common	EmfUtil.jar	EMF Functional Components shared lib
jars\common	ldapAuthModule.jar	Server API
jars\common	miconfig.jar	Configuration interfaces
jars\common	miserver.jar	TDI Server
jars\common	miggbl.jar	Library to migrate global.properties files
jars\common	mmconfig.jar	Configuration implementation
jars\common	tdiresource.jar	Server TMS implementation
jars\common	WebServiceUtil.jar	Web Services shared lib
jars\common	wsprov.jar	Web Services Provsioning
jars\common	wstrust.jar	Web Services Trust
jars\connectors	ACTConnector.jar	ACT Connector
jars\connectors	ADChangelogConnector.jar	Active Directory Changelog Connector
jars\connectors	ADChangelogv2Connector.jar	Active Directory Changelog (v.2) Connector

Table 1. (continued)

Location	Filename	Description
jars\connectors	ALConnector.jar	AssemblyLine (Iterator) Connector
jars\connectors	AxisEasyWSServerConnector.jar	Axis Easy Web Service Server Connector
jars\connectors	BTreeObjectDBConnector.jar	Btree Object DB Connector
jars\connectors	CommandLineConnector.jar	Command line Connector
jars\connectors	DBChangelogConnector.jar	RDBMS Changelog Connector
jars\connectors	DominoChangeDetectionConnector.jar	Domino Change Detection Connector
jars\connectors	DominoUsersConnector.jar	Domino Users Connector
jars\connectors	DSMLv2SOAPConnector.jar	DSMLv2 SOAP Connector
jars\connectors	DSMLv2SOAPServerConnector.jar	DSMLv2 SOAP Server Connector
jars\connectors	ExchangeChangelogConnector.jar	Exchange Changelog Connector
jars\connectors	FileSystemConnector.jar	File system Connector
jars\connectors	FTPClientConnector.jar	FTP Client Connector
jars\connectors	GLAConnector.jar	GLA Connector
jars\connectors	HTTPClientConnector.jar	HTTP Client Connector
jars\connectors	HTTPServerConnector.jar	HTTP Server Connector
jars\connectors	IBMMQConnector.jar	IBM MQ Connector
jars\connectors	IDSChangelogConnector.jar	IBM Directory Server Changelog Connector
jars\connectors	ITIMAgentConnector.jar	ITIM Agent Connector
jars\connectors	JDBCConnector.jar	JDBC Connector
jars\connectors	JMSConnector.jar	JMS Connector
jars\connectors	JMXConnector.jar	JMX Connector
jars\connectors	JNDIConnector.jar	JNDI Connector
jars\connectors	LDAPConnector.jar	LDAP Connector
jars\connectors	LDAPServerConnector.jar	LDAP Server Connector
jars\connectors	MailboxConnector.jar	Mailbox Connector
jars\connectors	MailboxConnectorUtils.jar	Mailbox Connector shared lib
jars\connectors	MemoryStreamConnector.jar	Memory Stream Connector
jars\connectors	MemQConnector.jar	Memory Queue Connector
jars\connectors	MQePasswordStoreConnector.jar	MQe Password Store Connector
jars\connectors	NetscapeChangelogConnector.jar	Netscape/iPlanet Changelog Connector
jars\connectors	NotesConnector.jar	Lotus Notes Connector
jars\connectors	OldHTTPClientConnector.jar	Old HTTP Client Connector
jars\connectors	OldHTTPServerConnector.jar	Old HTTP Server Connector
jars\connectors	PropertiesConnector.jar	Properties Connector
jars\connectors	RACConnector.jar	RAC Connector
jars\connectors	SapR3BorConnector.jar	Human Resources/Business Object Repository Connector for SAP R/3
jars\connectors	SapR3UserRegConnector.jar	User Registry Connector for SAP R/3
jars\connectors	ScriptConnector.jar	Script Connector
jars\connectors	ServerNotificationsConnector.jar	Server Notifications Connector

Table 1. (continued)

Location	Filename	Description
jars\connectors	SNMPConnector.jar	SNMP Connector
jars\connectors	SNMPServerConnector.jar	SNMP Server Connector
jars\connectors	SystemQueueConnector.jar	System Queue Connector
jars\connectors	SystemStoreConnector.jar	System Store Connector
jars\connectors	TAMConnector.jar	TAM Connector
jars\connectors	TCPConnector.jar	TCP Connector
jars\connectors	TCPServerConnector.jar	TCP Server Connector
jars\connectors	TimerConnector.jar	Timer Connector
jars\connectors	TIMDSMLv2Connector.jar	TIM DSMLv2 Connector
jars\connectors	URLConnector.jar	URL Connector
jars\connectors	WindowsUsersGroupsConnector.jar	Windows Users and Groups Connector
jars\connectors	WSReceiverServerConnector.jar	Web Service Receiver Server Connector
jars\connectors	zOSChangelogConnector.jar	z/OS Changelog Connector
jars\eventhandlers	ADLDAPSwitchboardEventHandler.jar	Active Directory Changelog EventHandler
jars\eventhandlers	ConnectorEventHandler.jar	Connector EventHandler
jars\eventhandlers	DSMLv2EventHandler.jar	DSMLv2 EventHandler
jars\eventhandlers	ExchangeLDAPSwitchboardEventHandler.jar	Exchange Changelog EventHandler
jars\eventhandlers	GenericThreadEventHandler.jar	Generic thread (primitive EventHandler)
jars\eventhandlers	HTTPEventHandler.jar	HTTP EventHandler
jars\eventhandlers	LDAPDEventHandler.jar	LDAP Server EventHandler
jars\eventhandlers	LDAPEventHandler.jar	LDAP EventHandler
jars\eventhandlers	MailboxEventHandler.jar	Mailbox EventHandler
jars\eventhandlers	SecureWayEventHandler.jar	IBM Directory Server EventHandler
jars\eventhandlers	SNMPEventHandler.jar	SNMP EventHandler
jars\eventhandlers	TCPEventHandler.jar	TCP Port EventHandler
jars\eventhandlers	TimerEventHandler.jar	Timer EventHandler (primitive EventHandler)
jars\eventhandlers	zOSLDAPEventHandler.jar	z/OS LDAP Changelog EventHandler
jars\functions	AssemblyLineFC.jar	AssemblyLine Function Component
jars\functions	AxisEasyInvokeSoapWSFC.jar	Axis EasyInvoke Soap WS FC
jars\functions	AxisJavaToSoapFC.jar	Axis Java To Soap FC
jars\functions	AxisSoapToJavaFC.jar	Axis Soap To Java FC
jars\functions	CastorJavaToXMLFC.jar	CastorJavaToXML FC
jars\functions	CastorXMLToJavaFC.jar	CastorXMLToJava FC
jars\functions	CBEGeneratorFC.jar	CBE Generator Function Component
jars\functions	ComplexTypesGeneratorFC.jar	Complex Types Generator FC
jars\functions	di_castor_mapping.xml	Defines how XML is mapped to Java objects by Castor
jars\functions	EmfSdoToXmlFC.jar	SDOToXML FC
jars\functions	EmfXmlToSdoFC.jar	XMLToSDO FC

Table 1. (continued)

Location	Filename	Description
jars\functions	InvokeSoapWSFC.jar	InvokeSoap WS FC
jars\functions	JavaClassFC.jar	Java Class Function Component
jars\functions	MemBufferQFC.jar	Memory Queue FC
jars\functions	ParserFC.jar	Parser FC
jars\functions	RemoteCmdLineFC.jar	Remote Command Line FC
jars\functions	SapR3RfcFC.jar	Function Component For SAP R/3
jars\functions	ScriptedFC.jar	Scripted FC
jars\functions	SendEMailFC.jar	SendEMail Function Component
jars\functions	WrapSoapFC.jar	WrapSoap FC
jars\functions	zOSTSOCommandLineFC.jar	z/OS TSO/E Command Line FC
jars\parsers	CSVParser.jar	CSV Parser
jars\parsers	DSMLParser.jar	DSML Parser
jars\parsers	DSMLv2Parser.jar	DSMLv2 Parser
jars\parsers	FixedParser.jar	Fixed Parser
jars\parsers	HTTPParser.jar	HTTP Parser
jars\parsers	LDIFParser.jar	LDIF Parser
jars\parsers	LineReaderParser.jar	Line Reader Parser
jars\parsers	ScriptParser.jar	Script Parser
jars\parsers	SimpleParser.jar	Simple Parser
jars\parsers	SOAPParser.jar	SOAP Parser
jars\parsers	SPMLv2Parser.jar	SPMLv2 Parser
jars\parsers	XMLParser.jar	XML Parser
jars\parsers	XMLSaxParser.jar	XML SAX Parser
jars\parsers	XSLbasedXMLParser.jar	XSL based XML parser
jars\plugins	idipwcrypto.jar	Plugin crypto utility
jars\plugins	mqeconfig.bat	MQe configuration utility script
jars\plugins	mqeconfig.jar	MQe configuration utility
jars\plugins	mqeconfig.props	MQe sample setup configuration

Solution Directory files

The following is a list of Solution Directory files: these files are automatically copied to the solution directory the first time the TDI server uses this solution directory.

Table 2.

File	Description
idisrv.sth	the TDI server stash file; it is a binary file, which contains the encrypted password for the sample server keystore file ("**testserver.jks**").
solution.properties	this file is a writable copy of the "global.properties" file and is used when the server is started from the solution directory; it is a text file; by default the file is in the platform native encoding.
testserver.jks	a binary file which contains the sample server certificate.

Table 2. (continued)

File	Description
etc/build.properties	this file contains the TDI Build information, build date, version, etc.; it is a text file, by default the file is in the platform native encoding.
etc/CSServersInfo.xml	an XML repository file for storing System Store configuration details.
etc/derby.properties	a text file, by default the file is in the platform native encoding. The default configuration file for the System Store shipped with TDI 6.1.1. For more details on this file refer to the Cloudscape™ v10.1 documentation.
etc/executetask.properties	this file controls the logging strategy of the TDI server (ibmdisrv) when started from the Config Editor (ibmditk). This is a log4j configuration file.
etc/global.properties	this file is a copy of the main configuration property file for TDI 6.1.1. This file in this location is not used by TDI itself.
etc/global.properties.v611	this is a placeholder for a sample file of TDI 6.1.1. This file is non-empty only if you have chosen to upgrade from previous versions of TDI to TDI 6.1.1. In such cases, since the file is a migrated file, it would be helpful for you to compare your migrated file to .v61 to see how it is different from doing a clean install. This file can be a helpful reference.
etc/ibmdi.ico	the icon file for TDI 6.1.1; it is a binary file.
etc/ITDIJ060100.sys	this is a product signature (license) file used by the ITLM agent to recognize TDI 6.1.1.
etc/jlog.properties	the default JLog configuration file for TDI 6.1.1. By default the jlog.properties file in the install folder is used. However, the JLog configuration file can be specified with the “jlog.configuration” system property. This property can be set in solution.properties, for example. That is why in order to use the jlog.properties in the solution folder, the “jlog.configuration” property in `solution.properties` must be set accordingly.
etc/log4j.properties	the Log4j configuration file for controlling the TDI 6.1.1 Server (ibmdisrv) logging. The configuration file can be specified with the “log4j.configuration” system property from the java command line in the TDI startup script.
etc/reconnect.rules	a text file (UTF-8 encoded) where you can define reconnect rules on how reconnect exceptions should be handled by TDI. Note that this file may contain language-specific characters. That is why it is important to keep this file UTF-8 encoded.
etc/tdisrvctl-log4j.properties	the Log4j configuration file for controlling the log traces generated by the tdisrvctl command line utility.
serverapi/cryptoutils.sh or serverapi/cryptoutils.bat	a command line utility (shell script) used for encrypting/decrypting TDI configurations and the user registry file.
serverapi/registry.txt	an example of a Server API user registry file; it is a text file in the platform native encoding.
serverapi/registry.enc	an encrypted version of the "registry.txt" (user registry); it is a binary file.
serverapi/testadmin.der	the exported certificate from "testadmin.jks"; it is a binary file.
serverapi/testadmin.jks	an example keystore/truststore file for a Server API remote client; it is a binary file.

Example Property files

An installation of IBM Tivoli Directory Integrator is to a large extent customized by means of a set of text files containing one of more **properties**, usually in the form of a keyword or identifier followed by a value. The following global property text files can be found at the `root/etc` level of the IBM Tivoli Directory Integrator installation directory:

- “log4j.properties” on page 32
- “ce-log4j.properties” on page 33

- "executetask.properties" on page 34
- "jlog.properties" on page 34
- "derby.properties" on page 35
- "global.properties" on page 36

Properties set in any of those files form a baseline for the entire IBM Tivoli Directory Integrator installation for all users on that machine. However, if your Solution Directory is different from the installation directory, you can have a set of text files in your Solutions Directory that mirror their counterparts in the installation directory. A property listed in any of those files will override anything set in any of the global installation property files mentioned above. Futhermore, a Java property set inside a Config file takes the highest precedence, and overrides anything in a global property file or the property files in the Solution Directory.

You can specify the Solution Directory in multiple ways:

- By setting the environment variable *TDI_SOLDIR* before starting the Config Editor or the Server
- By specifying the **-s** parameter to the `ibmditk` script or the `ibmdisrv` script to start respectively the Config Editor or the Server. This will take precedence over setting *TDI_SOLDIR*.

If *TDI_SOLDIR* equals the installation directory, the behavior will be like in older versions of TDI: all property files will be read from there, and the remarks about property files in the Solutions Directory do not apply.

In any other case, the first time you run the TDI Server, it will make a copy of all the property files into your Solutions Directory (it will not overwrite these files if they already exist). You can now tailor these files to your particular needs, without affecting the property files in the installation directory. The files remaining in the installation directory continue to form a baseline configuration for other instances of TDI.

Note: The file `global.properties` will be copied to a file called `solutions.properties` in your Solutions Directory. Other files, like `log4j.properties` and the files in the `amc` and `serverapi` folders will be copied under their own name.

log4j.properties

This file sets a baseline for the log-strategy for the server (ibmdisrv). See "executetask.properties" on page 34 for behavior of AssemblyLines run from the Config Editor (ibmditk) console, and "ce-log4j.properties" on page 33 for logging behavior of the Config Editor itself.

Log options configured in the Logging tab in the Config Editor are written into the Config file, and are supplementary to or supersede the following:

```
# This file controls the logging strategy for the server (ibmdisrv) when started
# from the command line.
# Look at executetask.properties for the logging strategy of the server when started
# from the Configuration Editor (ibmditk).
# Look at ce-log4j.properties for the logging behavior of the Configuration Editor (ibmditk).
#
# You will normally configure the logging strategy of the server by adding appenders
# using the Configuration Editor (ibmditk). This file only defines the baseline
# that is independent of the configuration files you are using.
#
# See the IDI documentation for more information on the contents of this file.
#


log4j.rootCategory=INFO, Default

# This is the default logger, you will see that it logs to ibmdi.log
log4j.appender.Default=org.apache.log4j.FileAppender
```

```
log4j.appender.Default.file=ibmdi.log
log4j.appender.Default.layout=org.apache.log4j.PatternLayout
log4j.appender.Default.layout.ConversionPattern=%d{ISO8601} %-5p [%c] - %m%n
log4j.appender.Default.append=false

# You may change the logging category of these subsystems to DEBUG
# if you want to investigate particular problems. This may
# generate a lot of output.
# ...com.ibm.di.config describes the loading of the configuration file (.xml),
#    and how the internal configuration structure is built.
# ...com.ibm.di.loader gives information about jar files, and where classes are found.
#    It also loads idi.inf files, which provides Connectors/Parsers/EH information
#    for the Configuration Editor.
log4j.logger.com.ibm.di.config=WARN
log4j.logger.com.ibm.di.loader=WARN

# Uncomment the lines below to activate them

# Here is an example on how to make a logger that logs to the console
#log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
#log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
#log4j.appender.CONSOLE.layout.ConversionPattern=%d [%t] %-5p - %m%n0

# Here is an example that logs to myFile.log
#log4j.appender.fileLOG=org.apache.log4j.FileAppender
#log4j.appender.fileLOG.file=myFILE.log
#log4j.appender.fileLOG.layout=org.apache.log4j.PatternLayout
#log4j.appender.fileLOG.layout.ConversionPattern=%d{ISO8601} %-5p [%c] - %m%n
#log4j.appender.fileLOG.append=false


# Finally, make use of the loggers defined above:
# Tell AssemblyLines myAL to log using CONSOLE logger defined above.

# log4j.logger.AssemblyLine.myAL=INFO, CONSOLE

# Or you could log to myFile.log

# log4j.logger.AssemblyLine.myAL=INFO, fileLOG

# Finally, make EventHandler myEH log to myFile.log
# log4j.logger.EventHandler.myEH=INFO, fileLOG
```

ce-log4j.properties

This file sets a baseline for the log-strategy for the Config Editor (ibmditk). See "executetask.properties" on page 34 for behavior of AssemblyLines run from the Config Editor (ibmditk) console.

```
# This file controls the logging strategy for the Configuration Editor (ibmditk)
# Look at log4j.properties for the logging strategy of the server when started from the command line.
#
# See the TDI documentation for more information on the contents of this file.
#


log4j.rootCategory=INFO, Default

# This is the default logger, you will see that it logs to ibmditk.log
log4j.appender.Default=org.apache.log4j.FileAppender
log4j.appender.Default.file=ibmditk.log
log4j.appender.Default.layout=org.apache.log4j.PatternLayout
log4j.appender.Default.layout.ConversionPattern=%d{ISO8601} %-5p [%c] - %m%n
log4j.appender.Default.append=false

# You may change the logging category of these subsystems to DEBUG
# if you want to investigate particular problems. This may
# generate a lot of output.
# ...com.ibm.di.admin gives logging from the Configuration Editor itself.
# ...com.ibm.di.config describes the loading of the configuration file (.xml),
#    and how the internal configuration structure is built.
# ...com.ibm.di.loader gives information about jar files, and where classes are found.
#    It also loads idi.inf files, which provides Connectors/Parsers/EH information
#    for the Configuration Editor.
log4j.logger.com.ibm.di.admin=INFO
log4j.logger.com.ibm.di.config=WARN
log4j.logger.com.ibm.di.loader=WARN
```

executetask.properties

This file controls the log-strategy for the Config Editor (ibmditk) console. See "log4j.properties" on page 32 for behavior of the server (ibmdisrv).

```
# This file controls the logging strategy of the server (ibmdisrv) when started
# from Configuration Editor (ibmditk).
# Look at log4j.properties for behavior of the server (ibmdisrv) when started
# from the command line.
# You will normally configure the logging strategy of the server by adding appenders
# using the Configuration Editor (ibmditk). This file only defines the baseline
# that is independent of the configuration files you are using.
#
# See the TDI documentation for more information on the contents of this file.

log4j.rootCategory=INFO, Default

log4j.appender.Default=org.apache.log4j.ConsoleAppender
log4j.appender.Default.layout=org.apache.log4j.PatternLayout
log4j.appender.Default.layout.ConversionPattern=%d{HH:mm:ss}  %m%n

log4j.logger.com.ibm.di.config=ERROR
log4j.logger.com.ibm.di.loader=ERROR
```

jlog.properties

This file configures the JLOG-based tracing and FFDC functionality of the TDI server. These values can be modified dynamically (during Server execution) using the LogCmd script if the property `jlog.noLogCmd` was set to **false** when the Server started.

Note: You would normally use log4j to trace execution flow in your solution; the JLOG-based tracing and FFDC is meant to aid IBM Support should you have problems with IBM Tivoli Directory Integrator.

```
###########################################################################################################
# This file controls the tracing and First Failure Data Capture (FFDC) strategy for TDI 6.0
# See the IDI documentation for more information on the contents of this file.
###########################################################################################################

#----------------------------------------------------------------------------------------------------------
# Enable the JLOG's command server
#
#  If the jlog.noLogCmd is set to false, then the JLOG LogManager will listen on the
#  default port (9992) for JLOG log commands.
#  Setting this property to false will enable you to modify the JLOG properties dynamically,
#  using the logcmd scripts. The logcmd scripts are placed in the TDI_HOME directory.
#  The default value is set to true.
#----------------------------------------------------------------------------------------------------------
jlog.noLogCmd=true

#----------------------------------------------------------------------------------------------------------
# Configure Jlog FileHandler for tracing into a file.
#
#  By default the FileHandler is not attached to the Jlog Logger.
# Uncomment the properties with the prefix jlog.filehandler below to configure a FileHandler.
# After uncommenting this you need to add the filehandler to the logger's listeners names as shown
# below
# e.g: jlog.logger.listenerNames=jlog.snapmemory jlog.snaphandler jlog.filehandler
#----------------------------------------------------------------------------------------------------------
#jlog.filehandler.className=com.ibm.log.FileHandler
#jlog.filehandler.description=JLOG File Handler for Logging and Tracing
#jlog.filehandler.encoding=UTF8
```

```
#jlog.filehandler.maxFiles=10
#jlog.filehandler.maxFileSize=2048
#jlog.filehandler.appending=true
#jlog.filehandler.fileDir=logs/
#jlog.filehandler.trace.fileName=trace.log
#-----------------------------------------------------------------------------------------------------------


#-----------------------------------------------------------------------------------------------------------
# create a level filter.
# The level filter is used to define the level at which JFFDC action will be triggered.
# For JFFDC to be meaningful this should be set to either FATAL or ERROR (case-insensitive).
# NOTE: Setting the trigger level to other levels such as DEBUG_MIN will trigger unwanted JFFDC
#  action causing a performance drop.
#-----------------------------------------------------------------------------------------------------------
jlog.levelflt.className=com.ibm.log.LevelFilter
jlog.levelflt.level=FATAL


#-----------------------------------------------------------------------------------------------------------
# Configure the SnapMemoryHandler for tracing into a memory buffer.
# The SnapMemoryHandler traces into a memory buffer and dumps the contents of the memory to a file on
# trigger of a event (as defined by the level filter above) and writes the content to the specified
# file
# Properties:
# jlog.snapmemory.queueCapacity : Sets the nnumber of LogEvents that can be buffered in the memory
# jlog.snapmemory.snapFile  : name of the file to which the contents of the memory will be dumped
# jlog.snapmemory.baseDir  : The directory where the snapFile is placed.
#        daily subdirectories will be created under this base directory, as:
#         [baseDir]/[YYYY-MM-DD]/
#         Note: MS-DOS style path names need to be be escaped with backslashes
#         eg: c:\\CTGI\\FFDC
# jlog.snapmemory.userSnapFile : The name of the file to which the user initiated (from logcmd) dumps
#        will be written to.
# jlog.snapmemory.userSnapDir  : The directory where the userSnapfile is placed.
# jlog.snapmemory.msgIds  : The list of TMS IDs
# jlog.snapmemory.msgIDRepeatTime   : The minimum time, in milliseconds, after passing a log event with a
#        given TMS message id, before another log event with the same id can
#        be passed.
#-----------------------------------------------------------------------------------------------------------
jlog.snapmemory.className=com.tivoli.log.SnapMemoryHandler
jlog.snapmemory.description=Memory handler used to trace to memory
jlog.snapmemory.queueCapacity=10000
jlog.snapmemory.dumpEvents=true
jlog.snapmemory.snapFile=trace.log
jlog.snapmemory.baseDir=CTGDI/FFDC/
jlog.snapmemory.userSnapFile=userTrace.log
jlog.snapmemory.userSnapDir=CTGDI/FFDC/user/
jlog.snapmemory.triggerFilter=jlog.levelflt
jlog.snapmemory.msgIds=*E
jlog.snapmemory.msgIDRepeatTime=10000

#-----------------------------------------------------------------------------------------------------------
# Configure the JLogSnapHandler taking a snapshot of the SnapMemoryHanlders buffer
# The JLogSnapHanlder takes a snapshot of the associated SnapMemoryBuffer.
#-----------------------------------------------------------------------------------------------------------
jlog.snaphandler.className=com.tivoli.log.JLogSnapHandler
jlog.snaphandler.description=snaphandler to dump the memory trace
jlog.snaphandler.baseDir=CTGDI/FFDC/
jlog.snaphandler.snapMemoryHandler=jlog.snapmemory
jlog.snaphandler.triggerFilter=jlog.levelflt

#-----------------------------------------------------------------------------------------------------------
# Configure the PDLogger (Problem Determination) Object and attach the Listeners to it.
# jlog.logger.level can be FATAL | ERROR | WARNING | INFO | DEBUG_MIN | DEBUG_MID | DEBUG_MAX
# The heirarchy of the log levels is from the most severe (FATAL) to the least severe (DEBUG_MAX)
# The value for this property is case-insensitive
#-----------------------------------------------------------------------------------------------------------
jlog.logger.level=FATAL
#jlog.logger.listenerNames=jlog.snapmemory jlog.snaphandler
jlog.logger.listenerNames=jlog.filehandler.trace
jlog.logger.className=com.ibm.log.PDLogger
```

derby.properties

This file contains some defaults for Cloudscape (Derby) in networked mode. Most TDI-related Cloudscape parameters are not maintained here but in `global.properties` and `solution.properties`. More information about these parameters can be obtained from the Cloudscape documentation.

```
# This is a sample properties file provided to show the proper format.
# We're also setting one property which will make sure that
# Cloudscape adds to the error log instead of overwriting it.
# This mode is useful for development.
derby.drda.logConnections=true
derby.drda.maxThreads=0
derby.drda.portNumber=1527
derby.drda.traceAll=true
derby.drda.timeSlice=0
derby.drda.traceDirectory=/trace
```

global.properties

This file is read by ibmditk and ibmdisrv on startup. This file is read and applied before a file called `solution.properties` from your Solution Directory is read (Note that the rendition here, due to extremely long line-lengths, may not be complete. Refer to an actual `global.properties` file instead):

```
##
## This file is read by ibmditk/ibmdisrv on startup
##
## Enter <name>=<value> to set system properties.
## Enter !include <file | url> to include other files
##

com.metamerge.securityTransformation=DES/ECB/NoPadding

##
## Modify the line below to add your own jar/zip files.
## The property may specify several directories or jar files, separated by the Java Property "path.separator",
## which is ":" on Linux and ";" on Windows
## Directories will be searched recursively by the TDILoader for jar files containing classes and resources.
## Only files with a ".zip" or ".jar" extension are searched.
# com.ibm.di.loader.userjars=c:\myjars

## Custom class for ibmjs options to let us choose regex library
ibmjs.options=com.ibm.di.script.ScriptEngineOptions

## Regex library selection (java or jakarta). Using jakarta requires the Jakarta regex library (not included)
com.ibm.di.scriptengine.regex=java

##
## Modify the line below to enable the config autoload feature. When this property is defined, the "ibmdisrv -d" command
## line will look for *.xml files in the directory specified by this property and start each one.
##
# com.ibm.di.server.autoload=autoload.tdi

##
## Modify the line below to specify the location of the Library Resources.
## The default value is <HOMEDIR>/tdilibrary
# com.ibm.di.admin.library.dir=

##
## SYSTEM STORE
##

## Location of the database (embedded mode) - Cloudscape 10
com.ibm.di.store.database=TDISysStore
com.ibm.di.store.jdbc.driver=org.apache.derby.jdbc.EmbeddedDriver
com.ibm.di.store.jdbc.urlprefix=jdbc:derby:
#com.ibm.di.store.jdbc.user=APP
#{protect}-com.ibm.di.store.jdbc.password=APP


## Location of the database to connect (networked mode) - Cloudscape 10 - DerbyClient driver
#com.ibm.di.store.database=jdbc:derby://localhost:1527/C:\Program Files\IBM\TDI\V6.1.1\TDISysStore;create=true
#com.ibm.di.store.jdbc.driver=org.apache.derby.jdbc.ClientDriver
#com.ibm.di.store.jdbc.urlprefix=jdbc:derby:
#com.ibm.di.store.jdbc.user=APP
#{protect}-com.ibm.di.store.jdbc.password=APP
#
## Details for starting Cloudscape in network mode.
## Note: If the com.ibm.di.store.hostname is set to localhost then remote connections will not be allowed.
## If it is set to the IP address of the local machine - then remote clients can access this Cloudscape
## instance by mentioning the IP address. The network server can only be started for the local machine.
#
#com.ibm.di.store.start.mode=automatic
#com.ibm.di.store.hostname=localhost
#com.ibm.di.store.port=1527
#com.ibm.di.store.sysibm=true


# the varchar(length) for the ID columns used in system store and pes connector tables
com.ibm.di.store.varchar.length=512
```

```
## create statements for system store tables (Cloudscape 5.1)
#com.ibm.di.store.create.delta.systable=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, SEQUENCEID int, VERSION int)
#com.ibm.di.store.create.delta.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, SEQUENCEID int, ENTRY long varbinary )
#com.ibm.di.store.create.property.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ENTRY long varbinary )
#com.ibm.di.store.create.checkpoint.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ALSTATE long varbinary, ENTRY long varbinary, TCB lo
#com.ibm.di.store.create.sandbox.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ENTRY long varbinary )

## create statements for system store tables (Cloudscape 10)
com.ibm.di.store.create.delta.systable=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, SEQUENCEID int, VERSION int);ALTER TABLE {0} ADD CONSTR
com.ibm.di.store.create.delta.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, SEQUENCEID int, ENTRY BLOB );ALTER TABLE {0} ADD CONSTRAIN
com.ibm.di.store.create.property.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ENTRY BLOB );ALTER TABLE {0} ADD CONSTRAINT IDI_PS_{UNI
com.ibm.di.store.create.checkpoint.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ALSTATE BLOB, ENTRY BLOB, TCB BLOB );ALTER TABLE {0}
com.ibm.di.store.create.sandbox.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ENTRY BLOB )
com.ibm.di.store.create.recal.conops=CREATE TABLE {0} (METHOD varchar(VARCHAR_LENGTH), RESULT BLOB, ERROR BLOB)


## create statements for system store tables DB2 on z/OS
#com.ibm.di.store.create.delta.systable=CREATE TABLESPACE TS1DSYS LOCKSIZE ROW BUFFERPOOL BP32K;CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL
#com.ibm.di.store.create.delta.store=CREATE TABLESPACE TS1DST LOCKSIZE ROW BUFFERPOOL BP32K;CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, SE
#com.ibm.di.store.create.property.store=CREATE TABLESPACE PS3DST LOCKSIZE ROW BUFFERPOOL BP32K;CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
#com.ibm.di.store.create.checkpoint.store=CREATE TABLESPACE TSPACE1 LOCKSIZE ROW BUFFERPOOL BP32K;CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NU
#com.ibm.di.store.create.sandbox.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ENTRY BLOB)
#com.ibm.di.store.create.recal.conops=CREATE TABLESPACE IM{UNIQUE} LOCKSIZE ROW BUFFERPOOL BP32K;CREATE TABLE {0} (METHOD VARCHAR(VARCHAR_LENGTH), RE
# Set a customized SQL statement for creation of the Tombstone Manager table. Keep the same table and field names.
#com.ibm.di.store.create.tombstones=CREATE TABLE IDI_TOMBSTONE ( ID INT GENERATED ALWAYS AS IDENTITY, COMPONENT_TYPE_ID INT, EVENT_TYPE_ID INT, START


# the ibmsnap_commitseq column name used by the RDBMS changelog connector
com.ibm.di.conn.rdbmschlog.cdcolname=ibmsnap_commitseq

## server authentication
## example
## javax.net.ssl.trustStore=d:\test\KeyRings\namtp2.jks
## javax.net.ssl.trustStorePassword=secret
## javax.net.ssl.trustStoreType=jks

javax.net.ssl.trustStore=
{protect}-javax.net.ssl.trustStorePassword=
javax.net.ssl.trustStoreType=

## client authentication
## example
## javax.net.ssl.keyStore=d:\test\KeyRings\namtp2.jks
## javax.net.ssl.keyStorePassword=secret
## javax.net.ssl.keyStoreType=jks

javax.net.ssl.keyStore=
{protect}-javax.net.ssl.keyStorePassword=
javax.net.ssl.keyStoreType=



## Turns on java debug
# javax.net.debug=true

## java interpreter override
# com.ibm.di.javacmd=
# com.ibm.di.installdir=

# Location of directory where the JRE TDI will use is installed
com.ibm.di.jvmdir=C:\Program Files\IBM\TDI\V6.1.1/jvm

## Limits the number of threads IDI uses
## Must be set higher than 3 to have any effect

# com.ibm.di.server.maxThreadsRunning=500

com.ibm.di.server.securemode=false
com.ibm.di.server.keystore=testserver.jks
com.ibm.di.server.key.alias=server


## Server API properties
## ---------------------

api.on=true
api.user.registry=serverapi/registry.txt
api.user.registry.encryption.on=false

api.remote.on=false
api.remote.ssl.on=true
api.remote.ssl.client.auth.on=true
api.remote.naming.port=1099
api.truststore=testserver.jks
{protect}-api.truststore.pass={encr}KJZq/75VbD18GS57eaQOJFm4tAEbVdc5HLwMLb6p8XzBm3L7Axnxurs+F1Bsk3Xdde32UcCnUmtywc+0r+SE2ES0LLD3T8sehGkJMKHkLKPqL4N0ltBX

## Specifies a list of IP addresses to accept non SSL connections from (host names are not accepted).
## Use space, comma or semicolon as delimiter between IP addresses. This property is only taken into account
## when api.remote.ssl.on is set to false.
## api.remote.nonssl.hosts=

api.jmx.on=false
api.jmx.remote.on=false

## The configuration files placed in this folder can be edited through the Server API.
## Configuration files placed in other folders cannot be edited through the Server API.
api.config.folder=C:\Program Files\IBM\TDI\V6.1.1\configs

## Timeout in minutes for configuration locks. A value of 0 means no timeout.
api.config.lock.timeout=0

## Specifies if the Server API methods for custom method invocation (Session.invokeCustom(...)) are allowed to be used.
## When api.custom.method.invoke.on is set to false and the Server API methods for custom method invocation are used,
## then an exception will be thrown.
## Only classes listed in api.custom.method.invoke.allowed.classes are allowed to be directly invoked.
## The default value is false.
api.custom.method.invoke.on=false

## Specifies the list of classes which can be directly invoked by the Server API methods for custom
## method invocation (Session.invokeCustom(...)).
## This property is only taken into account if api.custom.method.invoke.on is set to true.
## The classes in this list must be separated by a space, a comma or a semicolon.
## Example:
## api.custom.method.invoke.allowed.classes=com.ibm.MyClass,com.ibm.MyOtherClass
## In the above example only methods from the com.ibm.MyClass and com.ibm.MyOtherClass classes are
## allowed to be directly invoked.
api.custom.method.invoke.allowed.classes=

## api.custom.authentication points to a JavaScript text file that contains custom authentication code.
## For example: api.custom.authentication=ldap_auth.js.
## To enable the built-in LDAP Authentication mechanism, set this property to "[ldap]".
## For example: api.custom.authentication=[ldap]

##api.custom.authentication=[ldap]

## LDAP Authnetication properties
## ---------------------

## If this parameter is set to "true" and the LDAP Authnetication initialization fails, the whole Server API will not be started.
## If this parameter is missing or is set to "false" any LDAP Authentication initialization errors will be logged and the Server API will be started.
api.custom.authentication.ldap.critical=false

## LDAP Server hostname.
api.custom.authentication.ldap.hostname=

## LDAP server port number. For example, 389 for non-SSL or 636 for SSL.
api.custom.authentication.ldap.port=

## Specifies whether SSL is used to communicate with the LDAP Server.
## When set to "true" SSL will be used, otherwise SSL will not be used.
api.custom.authentication.ldap.ssl=

## Specifies the LDAP directory location where user searches will be preformed.
## When this property is not specified user searches will not be performed.
api.custom.authentication.ldap.searchbase=

## Specifies the user id attribute to be used in searches.
## When this property is not specified user searches will not be performed.
api.custom.authentication.ldap.userattribute=

## Specifies an LDAP Server administrator distinguished name that will be used for user searches.
## When this property is not specified anonymous bind will be used for user searches.
api.custom.authentication.ldap.admindn=

## Password for the LDAP Server administrator distinguished name.
{protect}-api.custom.authentication.ldap.adminpassword=


## Tombstone Manager properties
## ---------------------

com.ibm.di.tm.on=false
com.ibm.di.tm.autodel.age=0
com.ibm.di.tm.autodel.records.trigger.on=10000
com.ibm.di.tm.autodel.records.max=5000
com.ibm.di.tm.create.all=false


## ----------------------
## Help system properties
## ----------------------

## Name of help server, comment out if you want local help system
## The Tivoli library is at the following URL:
## http://publib.boulder.ibm.com/infocenter/tiv2help/index.jsp?toc=/com.ibm.IBMDI.doc_6.1.1/toc.xml
com.ibm.di.helpHost=idisurya.in.ibm.com

## Port for help system
com.ibm.di.helpPort=80

## --------------------------------------------
## AssemblyLinePool: Connector pooling defaults
## --------------------------------------------
##
## Note! These settings are only used when an AssemblyLine uses
## an AssemblyLinePool in combination with a Server mode connector.

## The number of seconds before a pooled connector times (e.g. is closed and no longer reused)
##      Less than zero means disable connector pooling
##  Zero means never timeout
##  Greater than zero sets the number of seconds before a connector is closed
com.ibm.di.server.connectorpooltimeout=42

## Comma separated list of connector interfaces that we never pool
com.ibm.di.server.connectorpoolexclude=com.ibm.di.connector.FileConnector,com.ibm.di.connector.ScriptConnector


## Properties for Windows IPv6 communications.
## Uncomment these properties for Windows IPv6 communication only.
## These properties will not affect IPv4 communication or IPv6 communication on Unices.
#java.net.preferIPv4Stack=false
#java.net.preferIPv6Addresses=true

## --------------------------------------------
## Performance settings
## --------------------------------------------
##
## Enable/Disable performance logging
com.ibm.di.server.perfStats=false

### ------------------------------------------
### Used by Config Report
###-------------------------------------------
### set this is you want to override the local language for Config Reports
# com.ibm.di.admin.configreport.translation=en

##----------------------
## System Queue settings
##----------------------
## If set to "true" the System Queue is initialized on startup and can be used;
## otherwise the System Queue is not initialized and cannot be used.
systemqueue.on=false

## Specifies the fully qualified name of the class that will be used as a JMS Driver.
# systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.IBMMQ
# systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.JMSScriptDriver
systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.IBMMQe

### MQe JMS driver initialization properties
## Specifies the location of the MQe initialization file.
## This file is used to initialize MQe on TDI server startup.
systemqueue.jmsdriver.param.mqe.file.ini=C:\Program Files\IBM\TDI\V6.1.1\MQePWStore\pwstore_server.ini

### MQ JMS driver initialization properties
# systemqueue.jmsdriver.param.jms.broker=<host:port>
# systemqueue.jmsdriver.param.jms.serverChannel=<channel_name>
# systemqueue.jmsdriver.param.jms.qManager=<queuemanger_name>
# systemqueue.jmsdriver.param.jms.sslCipher=<cipherSuite_name>
# systemqueue.jmsdriver.param.jms.sslUseFlag=false

### JMS Javascript driver initialization properties
## Specifies the location of the script file
# systemqueue.jmsdriver.param.js.jsfile=driver.js

## This is the place to put any JMS provider specific properties needed by a JMS Driver,
## which connects to a 3rd party JMS system.
## All JMS Driver properties should begin with the 'systemqueue.jmsdriver.param.' prefix.
## All properties having this prefix are passes to the JMS Driver on initialization after
## removing the 'systemqueue.jmsdriver.param.' prefix from the property name.
# systemqueue.jmsdriver.param.user.param1=value1
# systemqueue.jmsdriver.param.user.param2=value2
# ...

## Credentials used for authenticating to the target JMS system
# {protect}-systemqueue.auth.username=<username>
# {protect}-systemqueue.auth.password=<password>



## -----------------------------------------------
## Active Correlation Technology engine settings
## -----------------------------------------------

## Enable/disable the ACT engine feature of TDI.
# act.engine.on=true

## A name of a rule file for the ACT engine. The rules must be compiled.
# act.engine.rule.set.file=myrules.acts
```

Chapter 3. Supported platforms

The following are the platforms supported for IBM Tivoli Directory Integrator 6.1.1. For the latest list of platforms supported see: http://www-1.ibm.com/support/docview.wss?rs=697&uid=swg27008801.

This URL can also be used to see the platform supported by the TDI plugins and the bit support per platform.

Note: On all 64-bit operating systems, the TDI Server and CE run in 32-bit tolerance mode because it ships and uses a 32-bit JRE. This would be 31-bit tolerance on native z/OS or zSeries Linux.

- Microsoft Windows Intel IA32
  - Windows 2000 Professional (supported for application design, development, and testing only; no support for production use) (32bit supported)
  - Windows 2000 Server (32bit supported)
  - Windows 2000 Advanced Server (32bit supported)
  - Windows XP Pro (supported for application design, development, and testing only; no support for production use) (32bit supported)
  - Windows 2003 Standard Edition (32bit supported)
  - Windows 2003 Enterprise Edition (32bit supported)
- Microsoft Windows AMD64/EMT64
  - Windows Server 2003 Standard Edition (64bit supported)
  - Windows Server 2003 Enterprise Edition (64bit supported)
- AIX
  - AIX 5L 5.2 (32/64bit supported) (Recommended Maintenance package 5200-08 is required)
  - AIX 5L 5.3 (32/64bit supported) (Recommended Maintenance package 5300-03 is required)

  Note: Non-Admin installs are not supported on AIX.
- Sun Solaris SPARC
  - Solaris 9 (32/64bit supported) (Zones not supported)
  - Solaris 10 (32/64bit supported) (Zones not supported)
- HP-UX PA-RISC
  - HP-UX11iv2 (11.23) (32/64bit supported)
- HP-UX Itanium
  - HP-UX11iv2 (11.23) (32/64bit supported)
- Linux Intel IA32
  - RedHat Enterprise Linux ES/AS 3.0 (32bit supported)
  - RedHat Enterprise Linux ES/AS 4.0 (32bit supported)
  - SLES 9 (32bit supported)
  - SLES 10 (32bit supported)
  - Red Flag Data Center 5.0 SP1 / Asianix 2.0 SP1 (32bit supported)

  Note: A prerequisite on RedHat Enterprise Linux ES/AS and Red Flag Data Center 5.0 SP1 is that library package compat-libstdc++-296-2.96-132.7.2 or above is installed for the Installer to work.
- Linux AMD64/EMT64
  - RedHat Enterprise Linux ES/AS 4.0 (64bit supported)
  - SLES 9 (64bit supported)

  - SLES 10 (32bit supported)
- Linux on Power (pSeries, iSeries, OpenPower and JS20 Blades)
  - RedHat Enterprise Linux ES/AS 3.0 (64bit supported)
  - RedHat Enterprise Linux ES/AS 4.0 (64bit supported)
  - SLES 9 (64bit supported)
  - SLES 10 (32bit supported)

  Note: A prerequisite on RedHat Enterprise Linux ES/AS for this architecture is that library package compat-libstdc++-296-2.96-132.7.2 or above is installed for the Installer to work.
- Linux s/390 and zSeries®
  - RedHat Enterprise Linux ES/AS 3.0 (31bit supported)
  - RedHat Enterprise Linux ES/AS 4.0 (64bit supported)
  - SLES 9 (64bit supported)
  - SLES 10 (32bit supported)
- • Native s/390 and zSeries
  - z/OS 1.6, 1.7 and 1.8 (9/2006) — with limitations; refer to Chapter 14, “z/OS environment Support,” on page 167
- iSeries
  - i5/OS V5R4 and higher (64bit supported); the CE is not supported on this platform. In addition, the JVM and/or Embedded WAS Express used will be the system-installed one(s); no JVM or Embedded WAS Express will be installed as part of the TDI product install.

    The TDI server on i5/OS requires IBM Technology for Java Virtual Machine (also known as ″J2SE 5.0 32-bit JVM″ or ″IBM J9 VM″) be installed on the system

    If you decide to install AMC, WebSphere Application Server at minimum level 6.0 must be installed.

    The install can only be performed locally, as an Administrator, using a console install. No GUI install facility exists for i5/OS..

Chapter 4. Migrating from older versions

The IBM Tivoli Directory Integrator Installer can assist in migrating from TDI 6.0 and TDI 6.1 to TDI 6.1.1.

Migrating from IBM Tivoli Directory Integrator 6.0 to IBM Tivoli Directory Integrator 6.1.1

Do the following to migrate an IBM Tivoli Directory Integrator 6.0 system to IBM Tivoli Directory Integrator 6.1.1:

1. Save any configuration files created by IBM Tivoli Directory Integrator.

   Note: Sandbox data is version-specific; data recorded under any previous version will not play in version 6.1.1.
2. If you have installed IBM Tivoli Directory Integrator 6.0 as a Windows Service, uninstall the Service first.
3. Uninstall IBM Tivoli Directory Integrator 6.0
4. Install IBM Tivoli Directory Integrator 6.1.1
5. Copy your Config files and any other custom files, including Cloudscape databases from your old installation directory to the new installation directory. Version 6.1.1 supports a Solution Directory, and you may decide to copy the aforementioned Config files, property files, Cloudscape databases, etc. to this instead of to the installation directory of TDI version 6.1.1.

   Note: Optionally, you might decide to move the way your Delta information is maintained from the old Btree objects to Delta Tables in the System Store. The best strategy for doing this is engineering a situation where your Delta information is empty (for example, establishing a new baseline) and then switch from the Btree objects to the Cloudscape Delta Tables. Note that the parameter that used to hold the filename of the Btree objects now indicates a table name in a database, so some editing of this value might be required.
6. You can now safely remove the old installation directory.
7. EventHandlers are deprecated, and will be removed in a future version of IBM Tivoli Directory Integrator. They should be migrated to Server Mode Connectors; please see the section named "Migration from ChangeLog EventHandlers to ChangeLog Connectors" in *IBM Tivoli Directory Integrator 6.1.1: Reference Guide* for more information.
8. If you have used the ITIM Agent Connector in a previous version of TDI, you may need to change the way you configure SSL connections. The ITIM Agent Connector in TDI version 6.1.1 uses JSSE (Java based key store/trust store) for SSL authentication, and this requires that you configure the SSL related certificate details in the `global.properties` or `solution.properties` file; instead of mentioning the certificate name in the old ITIM Agent Connector's ″CA Certificate File″ Parameter. These are the steps involved:
   a. Import the ITIM Agent's certificate that was previously mentioned in the ″CA Certificate File″ parameter into the TDI Trust store with for example *keytool* (iKeyman can be used too):

      ```
      keytool –import –file servercertificate.der –keystore tim.jks
      ```

      In this example the truststore is stored in the file `tim.jks`.
   b. Configure this truststore in the ″server authentication″ section of the `global.properties` or `solution.properties` file:

```
## server authentication

javax.net.ssl.trustStore=E:\IBMDirectoryIntegrator\tim.jks
{protect}-javax.net.ssl.trustStorePassword=<jks_keystore_password>
javax.net.ssl.trustStoreType=jks
```

Now, the ITIM Agent Connector will use the same JSSE-based secure communications architecture as the rest of TDI.

9. If you have used any of the Castor-based Function Components in your Configs, you will need to remove those and port their usage to the new SDO-based ones. A few words on this migration can be found in the *IBM Tivoli Directory Integrator 6.1.1: Reference Guide*, in the section "Function Components>XMLTOSDO FC>Migration".
10. Determine if multiple instances of AssemblyLines in different Java VMs attempt to use the same System Store, either for Checkpoint/Restart information or for Persistent Objects like Delta Tables. If so, configure the System Store (Cloudscape) for network access, as outlined in Chapter 7, “System Store,” on page 91.
11. IBM Tivoli Directory Integrator now requires and includes a J2SE version 1.5 compliant JVM (J2SE version 1.5 SR3). If you have developed your own code in Java, linked this code against the JVM libraries and integrated this with your IBM Tivoli Directory Integrator solution, you might need to recompile and re-link your code.
12. Previous versions of TDI used the (MY)CLASSPATH variable in the ibmditk and ibmdisrv command line scripts; the current version has the required path information built in and does not require this variable anymore. If you had tailored the aforementioned scripts before to include some libraries of your own, you don’t need to do anything with the CLASSPATH variable; just make sure your library is in the correct place (typically in the `jars/` directory) so it will be found by TDI. Alternatively, use the new `com.ibm.di.loader.userjars` property in `global.properties` to point to your own directory to be included in the loader path.
13. Test your solution with the new version.
14. If required, install the Windows Service.

Migrating from IBM Tivoli Directory Integrator 6.1 to IBM Tivoli Directory Integrator 6.1.1

Upgrades from TDI 6.1 to TDI 6.1.1 are accomplished automatically by means of the built-in update mechanisms of the System Installer (SI). Notably,

- IEHS 3.1.1: The installer will migrate existing TDI 6.1 installations from IEHS 3.0.1 up to 3.1.1 if IEHS was installed.
- IBM JRE 5.0 SR3: The installer will migrate existing TDI 6.1 installations from IBM JRE 5.0 SR1 up to IBM JRE 5.0 SR3 if any component requiring IBM JRE 5.0 SR1 was installed.
- Modifications to global.properties files: there are some new properties (ACT Engine settings); entries for these properties are added. Semantics of some System Store DDL statements have changed; properties defining them will be amended accordingly.
- Modifications to am_config.properties files: new property specifying AM update thread frequency.

On new (non-upgrade) installations, the newest versions of these components are installed.

The following commands are used during the migration of TDI 6.1 to the current release. They are being left after the install because they may be useful to support and customers.

tdimiggbl – For windows the file name is tdimiggbl.bat

This script is used to migrate any global.properties file starting with TDI 6.1 to the current release. Logging for this command is controlled by the tdimiggbl-log4j.properties file.

The usage for this command is:

```
tdimiggbl –f propfile [-b backfile] [-n newfile] [-v] [-?]
where:
-f propfile - The name of the file to migrate
-b backfile - Backup the original file with the specified name
-n newfile - Name to give the file that is migrated
-v            - Enable verbose mode
-?            - Prints the usage statement
```

tdimigam – For windows the file name is tdimigam.bat

This script is used to migrate any am_config.properties file starting with TDI 6.1 to the current release. Logging for this command is controlled by the tdimigam-log4j.properties file.

The usage for this command is:

```
tdimigam –f propfile [-b backfile] [-n newfile] [-v] [-?]
where:
-f propfile - The name of the file to migrate
-b backfile - Backup the original file with the specified name
-n newfile - Name to give the file that is migrated
-v            - Enable verbose mode
-?            - Prints the usage statement
```

Chapter 5. Security and TDI

Introduction

Security features are found throughout IBM Tivoli Directory Integrator (TDI). Some features secure access into remote systems from TDI, others protect access into TDI from remote systems, and yet others provide mechanisms to secure data, such as user credentials into remote systems.

Many of the features described in this chapter are not necessary when running TDI in a stand-alone mode in a secured environment. However, the features come in handy when other systems need to communicate with TDI, such as through the remote Web Admin Console (AMC) management tool or the TDI Remote Server API. Furthermore, if multiple people have access to the TDI server it could be necessary to protect access to confidential data, as well as maintain the integrity of the integration rules that TDI executes.

This chapter will explain the following features:

1. "Secure Sockets Layer (SSL) Support"
2. "Remote Server API" on page 56
3. "TDI Server Instance Security" on page 73
4. "Miscellaneous Config File features" on page 77
5. "Web Admin Console Security" on page 80
6. "Summary of configuration files and properties dealing with security" on page 80
7. "Miscellaneous security aspects" on page 81

This guide does not describe all the security capabilities of the individual TDI Components. Some common elements are described in "Miscellaneous security aspects" on page 81, however for individual elements of security configuration in the individual TDI components, consult the *IBM Tivoli Directory Integrator 6.1.1: Reference Guide*.

Secure Sockets Layer (SSL) Support

As you will see in this section, SSL is an important foundation for many of the security features. A working level knowledge of SSL should be acquired to fully exploit the capabilities in TDI.

The following Connectors and EventHandlers support SSL with properly configured IBM Tivoli Directory Integrator Servers:

- Connectors
  - JMS Connector
  - LDAP Connector
  - LDAP Server Connector
  - Netscape/iPlanet LDAP ChangeLog Connector
  - IBM Directory Server ChangeLog Connector
  - Active Directory ChangeLog Connector
  - Lotus® Notes® Connector
  - Axis Easy Web Service Server Connector
  - Web Service Receiver Server Connector
  - DSMLv2 SOAP Server Connector
- EventHandlers

- Active Directory ChangeLog EventHandler
- DSMLv2 EventHandler
- Exchange ChangeLog EventHandler
- HTTP EventHandler
- LDAP EventHandler
- LDAP Server EventHandler
- IBM Directory Server EventHandler
- Web Service EventHandler
- z/OS® LDAP EventHandler

SSL provides for encryption and authentication of network traffic between two remote communicating parties. Most production deployments of TDI make use of SSL. That is why SSL support is one of the major security features of TDI. More information on SSL as well as information on using SSL in Java programs from a development point of view can be found at http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html

TDI can be used as a client, as a server or as both at the same time. Configuring TDI for SSL when used as a client is different from configuring TDI when used as a server. That is why this section has been divided in two sub-sections – SSL for the server side and SSL for the client side.

Server SSL configuration of TDI components

When a TDI component is used as a server (for example a Server mode Connector) SSL mandates that a keystore to be used by TDI must be defined. For information on keystores and truststores please see the guide at http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html. The following steps are required to enable SSL support for IBM Tivoli Directory Integrator as a server:

1. If you don't have a java (jks) keystore file already in IBM Tivoli Directory Integrator create a keystore file using *keytool* (found in *root_directory*/jvm/jre/bin, or *(root_directory*/jvm/bin depending on your platform). If you don't have a personal key to be used in IBM Tivoli Directory Integrator get one from a Certificate Authority or create a self-signed key.
2. If the certificate in the IBM Tivoli Directory Integrator is a self-signed certificate, export the certificate.
3. If the IBM Tivoli Directory Integrator certificate is a self-signed certificate, using a key tool, import the exported IBM Tivoli Directory Integrator certificate to the keystore file in the client as a root authority certificate.
4. Edit *root_directory*/global.properties file for the keystore file location, keystore file password and keystore file type. In the current release, we support jks-type only.

   ```
   # Keystore file information for the server
     IBM Tivoli Directory Integrator authentication.
   # It is used to provide the public key of the IBM Tivoli Directory Integrator
    to the SSL enabled client.
   javax.net.ssl.keyStore=D:\test\clientStore.jks
   javax.net.ssl.keyStorePassword=secret
   javax.net.ssl.keyStoreType=jks
   ```

5. Edit *root_directory*/jvm/lib/security/java.security for the security provider list.

   ```
   security.provider.1=com.ibm.jsse.IBMJSSEProvider
   security.provider.2=com.ibm.crypto.provider.IBMJCE
   security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
   security.provider.4=com.ibm.security.cert.IBMCertPath

   #
   # SSLServerSocketFactory Provider
   #
   ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
   ```

6. Enable SSL for the clients (for example, using https in the Web browser).
7. Restart IBM Tivoli Directory Integrator

Notes:

1. The TDI server does not manage the keystores/truststores. All that the TDI server provides to the TDI components in terms of keystore support is the `global.properties` or `solution.properties` files, in which the standard Java keystore/truststore properties can be specified.
2. A TDI component can choose to use the default configured keystore/truststore in `global.properties` or `solution.properties`, or it can choose to implement its own handling of SSL sockets (for example implementing a custom SSLServerSocket Java class) so that it can use keystores/truststores different from the default.
3. If TDI needs to use both a client and a server certificate only the default certificate configured in `global.properties` or `solution.properties` is used, then this must be the same certificate. An alternative would be to write a custom implementation of the SSLSocket or the SSLServerSocket Java class and make it use a certificate different from the default.
4. See section "Certificates for the TDI Web Service Suite" on page 81 for specifics on the certificates for TDI Web Service components.

Client SSL configuration of TDI components

When a TDI component is used as a client (for example the LDAP Connector) SSL mandates that a truststore to be used by TDI must be defined. For information on keystores and truststores please see the guide at http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html.

The following steps are required to enable SSL support for IBM Tivoli Directory Integrator as a client:

1. Configure a server (such as IBM Tivoli Directory Server) to enable SSL.
2. If the certificate in the server is a self-signed certificate, export the certificate.
3. If you don't have a Java (jks) keystore file already, create a keystore file using *keytool* (found in *root_directory*/jvm/jre/bin, or *root_directory*/jvm/bin, depending on your platform) for IBM Tivoli Directory Integrator.
4. If the server certificate is a self-signed certificate, import the server certificate to the IBM Tivoli Directory Integrator keystore file as a root authority certificate using keytool.
5. Edit *root_directory*/etc/global.properties file for the keystore file location, keystore file password and keystore file type. In the current release, we support jks-type only.

   ```
   # Keystore file information for the server authentication.
   # It is used to verify the server's public key.
   # example
   javax.net.ssl.trustStore=d:\test\KeyRings\ibmdi.jks
   javax.net.ssl.trustStorePassword=secret
   javax.net.ssl.trustStoreType=jks

   # Keystore file information for the client authentication.
   # It is used to provide the public key of the
    IBM Tivoli Directory Integrator to the server
    if the server requests the client authentication.
   # example
   javax.net.ssl.keyStore=d:\test\KeyRings\ibmdi.jks
   javax.net.ssl.keyStorePassword=secret
   javax.net.ssl.keyStoreType=jks
   ```

6. Edit *root_directory*/jvm/lib/security/java.security for the security provider list.

   ```
   security.provider.1=com.ibm.jsse.IBMJSSEProvider
   security.provider.2=com.ibm.crypto.provider.IBMJCE
   security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
   security.provider.4=com.ibm.security.cert.IBMCertPath
   ```

7. Enable SSL for the Connectors.
8. Restart IBM Tivoli Directory Integrator

Note: TDI truststore and keystore do not play any part in SSL configuration for the Domino Change Detection connector. See section "Lotus Domino SSL specifics" on page 81 for more information.

SSL client authentication

If a TDI component is used as a client and the server with which it communicates requires SSL client authentication, then apart from a truststore TDI will need a keystore as well. In this case the keystore can be defined just like it is defined when TDI is used a server – please see the section "Server SSL configuration of TDI components" on page 48.

Note: Client TDI components which support SSL client authentication do not normally need a "SSL client authentication" check-box as do TDI server components. All such a client TDI component needs in order to prove its identity to the server is to have its keystore generated and configured in `global.properties` or `solution.properties`. If the server requires an SSL client certificate then the client SSL library will automatically send the client's certificate from the keystore configured in `global.properties` or `solution.properties`.

IBM Tivoli Directory Integrator and Microsoft Active Directory SSL configuration

Do the following to configure SSL for IBM Tivoli Directory Integrator and Microsoft Active Directory:

1. Install Certificate Services on Windows 2000 Server and an Enterprise Certificate Authority in the Active Directory Domain. Details are available at http://www.ntfaq.com/Articles/Index.cfm?ArticleID=14923. Make sure you install an **Enterprise Certificate Authority**.
2. Start the Certificate Server Service. This creates a virtual directory in Internet Information Service (IIS) that enables you to distribute certificates.
3. Create a Security (Group) Policy to direct Domain Controllers to get an SSL certificate from the Certificate Authority (CA).
   a. Open the **Active Directory Users and Computers Administrative** tool.
   b. Under the domain, right-click on **Domain Controllers**.
   c. Select **Properties**.
   d. In the **Group Policy** tab, click to edit the **Default Domain Controllers Policy**.
   e. Go to **Computer Configuration–>Windows Settings–>Security Settings–>Public Key Policies**.
   f. Right click **Automatic Certificate Request Settings**.
   g. Select **New**.
   h. Select **Automatic Certificate Request**.
   i. Run the wizard. Select the **Certificate Template for a Domain Controller**.
   j. Select your **Enterprise Certificate Authority** as the CA. Selecting a third-party CA works as well.
   k. Complete the wizard.
   l. All Domain Controllers automatically request a certificate from the CA, and support LDAP using SSL on port 636.
4. Retrieve the Certificate Authority Certificate to the machine on which you installed IBM Tivoli Directory Integrator.

   Note: You must install IIS before installing the certificate server.

   a. Open a Web browser on the machine on which you installed IBM Tivoli Directory Integrator.
   b. Go to http://*server_name*/certsrv/(where *server_name*is the name of the Windows 2000 server). You are asked to log in.
   c. Select the task **Retrieve the CA certificate or certificate revocation list**.
   d. Click **Next**.
   e. The next page automatically highlights the CA certificate. Click **Download CA certificate**.
   f. A new download window opens. Save the file to the hard drive.
5. Create a certificate store using keytool. Use keytool.exe to create the certificate store and import the CA certificate into this store.

Note: Keytool.exe is located in the IBM Tivoli Directory Integrator directory under /jvm/jre/bin Use the following command:

```
jvm\jre\bin\keytool -import -file
 certnew.cer -keystore keystore_name.jks
 -storepass password-alias keyalias_name
```

For example, assume the following values:

```
Keystorename = idi.jks
Password = secret
Keyalias name = AD_CA
```

The command looks resembles this script:

```
C:\Program Files\IBM\TDI\V6.1.1\jvm\jre\bin\keytool -import
-file certnew.cer -keystore idi.jks  -storepass secret -alias AD_CA
```

To verify the contents of your keystore, type the following script:

```
C:\Program Files\IBM\TDI\V6.1.1\jvm\jre\bin\keytool
 -list -keystore idi.jks -storepass secret
```

The following lines result:

```
Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry:

ad_ca, Mon Nov 04 22:11:46 MST 2002, trustedCertEntry,
Certificate fingerprint (MD5): A0:2D:0E:4A:68:34:7F:A0:21:36:78:65:A7:1B:25:55
```

6. Configure IBM Tivoli Directory Integrator to use the keystore created in the previous step. Edit *root_directory*/global.properties file for the keystore file location, keystore file password and keystore file type. In the current release, only jks-type is supported.

```
#server authentication
#example
javax.net.ssl.trustStore=c:\test\idi.jks
javax.net.ssl.trustStorePassword=secret
javax.net.ssl.trustStoreType=jks
#client authentication
#example
javax.net.ssl.keyStore=c:\test\idi.jks
javax.net.ssl.keyStorePassword=secret
javax.net.ssl.keyStoreType=jks
```

7. Edit *root_directory*/jvm/lib/security/java.security for the security provider list:

```
security.provider.1=com.ibm.jsse.IBMJSSEProvider
security.provider.2=com.ibm.crypto.provider.IBMJCE
security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
security.provider.4=com.ibm.security.cert.IBMCertPath
```

8. Enable SSL for your LDAP connector.
   a. Go to the JLDAP Connector or LDAP Connector configuration window.
   b. Change **LDAP URL** to port 636.
   c. Check **Use SSL**.
9. Restart IBM Tivoli Directory Integrator.

Note: The TDI Windows service wrapper permits you to start TDI as multiple service instances.

Self-signed vs. CA-signed certificates

CA-signed certificates

Typically there is a local certification authority (CA), i.e. the certificates will not come from any of the well known CAs like VeriSign, etc. The local CA itself should have a root certificate issued by a well known CA, but even this is not always true. If the local CA's root certificate is self-signed, you must import it into the trust store of each server or client that is using SSL. Please see section "Keystore and truststore management" on information how to do this. Each server for an SSL connection and each client doing PKI authentication must then issue a request for a certificate to the local CA, and must add the resulting certificate into its keystore.

Self-signed certificates

In this case, each server for an SSL connection, and each client doing PKI authentication, generates its own self-signed certificate. It is then necessary to export the certificate to a file and to import it into various trust stores. If a client **C** connects to a server **S**, C must have S's self-signed certificate in its trust store. If a client C does PKI authentication (symmetric SSL) to a server S, S must have C's self-signed certificate in its trust store. Note: Self-signed certificates can be used for either a client or a server certificate.

Keystore and truststore management

In order to create, edit, export and overall manage keystores and truststores the "iKeyman" GUI utility or the "keytool" command line utility can be used. The executables for these two utilities can be found in the "<TDI_INSTALL_FOLDER>" folder.

Managing a CA-signed certificate using keytool

Normally the process of acquiring and using CA-signed certificates goes like this: First a key pair is generated. After that a certificate for the public key is requested from a Certification Authority. When the Certification Authority sends back the signed certificate, the certificate is imported into the appropriate truststore. Below is an example of this process using the *keytool* Java utility:

1. `keytool -genkey -dname cn=<server_ip_address> -keystore server.jks -storepass secret -keypass secret`

   This command creates a private/public key pair and stores it into the 'server.jks' keystore file.

2. `keytool –certreq –file myRequest.csr –keystore server.jks –storepass secret –keypass secret`

   This command creates a Certificate Signing Request in the 'myRequest.csr' file for the public key created in step 1. The created Certificate Signing Request now can be sent to a Certification Authority.

3. `keytool -import -trustcacerts -file server.cer -keystore mytruststore.jks -storepass secret`

   This command reads the certificate (public key) stored in the 'server.cer' certificate file (possibly the response of a Certificate Signing Request) and imports it into the 'mytruststore.jks' keystore file.

After these steps are executed, the keystore 'server.jks' contains the created public/private key pair. On the other hand, the 'mytruststore.jks' keystore represents a truststore which trusts the public key.

Creating a self-signed certificate using keytool

The standard Java utility *keytool* can be used to create self-signed certificates. The following commands can be entered at the command prompt in the JVM bin directory:

1. `keytool -genkey -dname cn=server_ip_address -validity 18263 -keystore server.jks -storepass secret -keypass secret`

   This command creates a private/public key pair and stores it into the //server.jks keystore file.

2. `keytool -export -alias mykey -file server.cer -keystore server.jks -storepass secret`

   This command extracts the public key (certificate) from the server.jks //keystore file and stores it into the server.cer certificate file.

3. `keytool -import -trustcacerts -file server.cer -keystore server.jks -storepass secret -alias mytrustedkey`

This command reads the certificate (public key) stored in the server.cer //certificate file and imports it into the server.jks keystore file.

Answer "yes" and press Enter to the ″Certificate already exists in keystore under alias <mykey> Do you still want to add it? [no]:″ question.

Note: The number 18263 is the validity period of the certificate in days (18263 days is roughly equal to 50 years).

After these steps are executed a server.jks keystore file is created which contains a key for the server. This file is also a truststore which contains the server public key, i.e. trusts the server key. In this way this server.jks file can be used as both the server keystore and client truststore file.

Creating a self-signed certificate using iKeyman

iKeyman provides a graphical user interface for managing keystores and truststores. The iKeyman tool can be launched from within the TDI Config Editor by choosing “KeyManager” from the “Tools” menu. In order to create a keystore which contains a self-signed certificate you have to follow these steps:

1. Start the iKeyman GUI tool
2. From the **Key Database File** menu click **New...**.
3. In the **New** dialog box:
   a. set the **Key database type** to **JKS** (the default)
   b. set the **File Name** to `server.jks`
   c. set the **Location** to the appropriate value
4. In the **Password Prompt** dialog box:
   a. enter the keystore password you have chosen
   b. confirm the password value
   c. click **OK**
5. From the **Create** menu, click **New Self-Signed Certificate...**.
6. In the **Create New Self-Signed Certificate** dialog box:
   a. Set the **Key Label** to a name (label) that is used to identify the key and certificate in the database, for example, `my self-signed certificate`
   b. Set the **Key Size**
   c. Set the **Common Name** to the fully qualified host name of the server as the common name, for example, `www.myserver.com`
   d. Set the **Organization** to your organization’s name
   e. Fill in any of the optional fields if you need to
   f. Specify the **Country or region**
   g. Specify the **Validity Period** in days.
   h. Click **OK**
7. Click the **Extract Certificate...** button.
8. In the **Extract Certificate to a File** dialog box:
   a. Set the **Data type** to `Base64-encoded ASCII data` (the default)
   b. Set the **Certificate file name** to `server.arm`
   c. Set the “Location” to the appropriate value.
   d. Click **OK**
9. From the drop down list select **Signer Certificates**
10. Click the **Add...** button.
11. In the **Add CA’s Certificate from a File** dialog box:
    a. For the **Data Type** select **Base64-encoded ASCII data**
    b. Set the **Certificate file name** to `server.arm`

c. Input the **Location**.
d. Click **OK**.

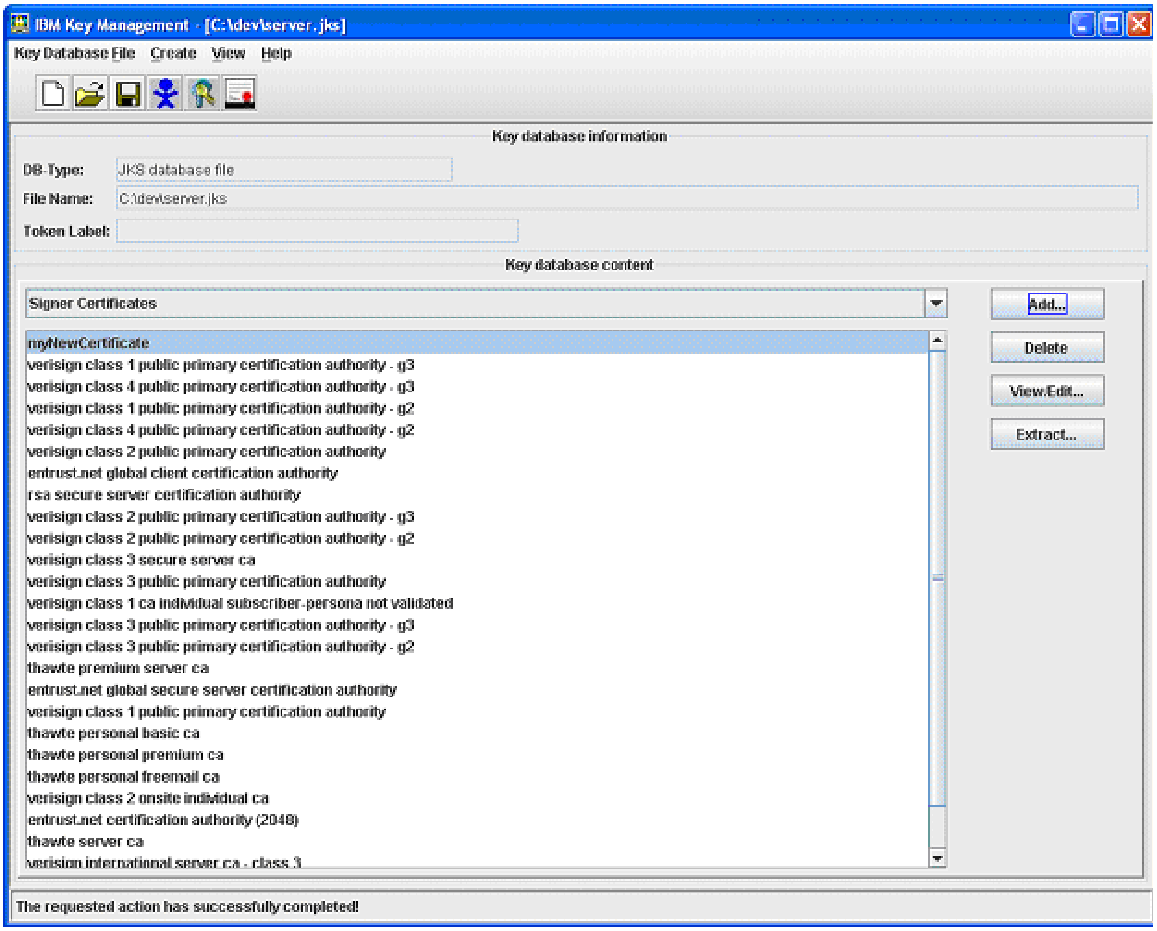
After these steps are executed a server.jks keystore file is created which contains a key for the server. This file is also a truststore which contains the server public key, i.e. trusts the server key. In this way this server.jks file can be used as both the server keystore and client truststore file.

Exporting a key from a keystore to a PKCS#12 file using iKeyman

1. Start the iKeyman GUI tool
2. From the **Key Database File** menu click **Open...**.
3. In the **Open** dialog select the path to the keystore. Make sure the **Key database type** is set properly.
4. In the combo box in the **Key database content** section select **Personal Certificates**.
5. Now all keys available in the keystore should be displayed in the **Key database content** section.
6. Select one of them and click the **Export/Import...** button.
7. In the **Export/Import...** dialog box:
   a. Select the **Export Key** radio button.
   b. Select **PKCS12** in the **Key file type** combo box.
   c. Enter file name and location where the PKCS12 file will be created.
   d. Click **OK**.

Importing a key from a PKCS#12 file into a keystore using iKeyman

1. Start the iKeyman GUI tool
2. From the **Key Database File** menu click **Open...**
3. In the **Open** dialog select the path to the keystore. Make sure the **Key database type** is set properly.
4. In the combo box in the **Key database content** section select **Personal Certificates**.
5. Now all keys available in the keystore should be displayed in the **Key database content** section.
6. Click the **Export/Import...** button.
7. In the **Export/Import...** dialog box:
   a. Select the **Import Key** radio button.
   b. Select **PKCS12** in the **Key file type** combo box.
   c. Enter file name and location of the PKCS12 file, which contains the key to be imported.
   d. Click **OK**.



DB-Type: JKS database file

File Name: C:\dev\server.jks

Token Label:

Signer Certificates

myNewCertificate
verisign class 1 public primary certification authority - g3
verisign class 4 public primary certification authority - g3
verisign class 1 public primary certification authority - g2
verisign class 4 public primary certification authority - g2
verisign class 2 public primary certification authority
entrust.net global client certification authority
rsa secure server certification authority
verisign class 2 public primary certification authority - g3
verisign class 2 public primary certification authority - g2
verisign class 3 secure server ca
verisign class 3 public primary certification authority
verisign class 1 ca individual subscriber-persona not validated
verisign class 3 public primary certification authority - g3
verisign class 3 public primary certification authority - g2
thawte premium server ca
entrust.net global secure server certification authority
verisign class 1 public primary certification authority
thawte personal basic ca
thawte personal premium ca
thawte personal freemail ca
verisign class 2 onsite individual ca
entrust.net certification authority (2048)
thawte server ca
verisign international server ca - class 3

The requested action has successfully completed!

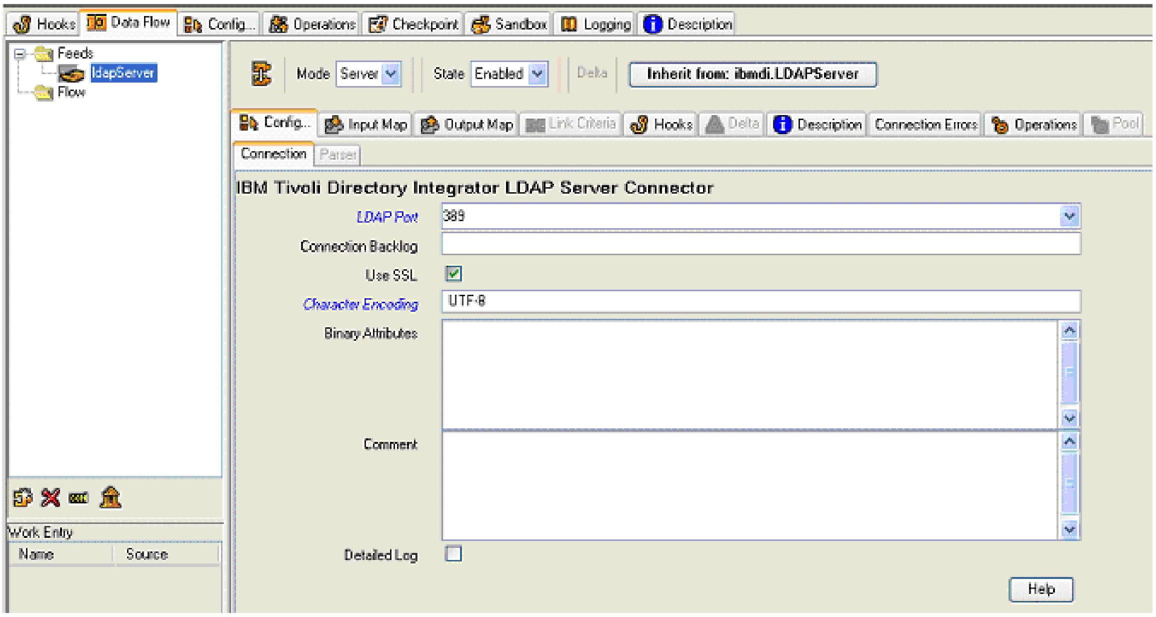
SSL example

In order to demonstrate how TDI can be configured for SSL when used as a server and also when used as a client, two examples are provided – one deploying the LDAP Server Connector and one deploying the LDAP Connector.

TDI component as a server

This example uses the LDAP Server Connector. The LDAP Server Connector listens for LDAP requests. When an LDAP request arrives the Connector parses the request and provides the request data to the hosting AssemblyLine. The AssemblyLine then processes the request and provides the data for the response to the LDAP Server Connector, so that it can build the LDAP response and send it back to the LDAP client. What follows is a step by step guide how to configure TDI for SSL when the LDAP Server Connector is used:

1. Obtain the server keystore either requesting it from a Certification Authority (CA) or creating a self-signed certificate as explained in either the““Creating a self-signed certificate using keytool” on page 52” section or the ““Creating a self-signed certificate using iKeyman” on page 53” section.
2. Set the keystore details in `global.properties` or `solution.properties` as described in the ““Server SSL configuration of TDI components” on page 48” section.
3. Check the “Use SSL” check-box on the Connector GUI configuration panel.



Mode Server
State Enabled
Inherit from: ibmdi.LDAPServer

IBM Tivoli Directory Integrator LDAP Server Connector

LDAP Port 389
Connection Backlog
Use SSL ☑
Character Encoding UTF-8
Detailed Log □

TDI component as a client

This example uses the LDAP Connector. The LDAP Connector connects to an LDAP Server and sends an LDAP request. After the Server returns the LDAP response the LDAP Connector provides that response to the AssemblyLine for further processing. What follows is a step by step guide how to configure TDI for SSL when the LDAP Connector is used:

1. Generate the client truststore.
2. Import the LDAP server certificate into the client truststore.
3. Set the truststore details in `global.properties` or `solution.properties` as described in the ""Client SSL configuration of TDI components" on page 49" section.

The following command line imports an existing certificate into a keystore (the keystore is created if not already existing):

```
keytool -import -trustcacerts -file myLDAPServerCert.cer -keystore myClientTruststore.jks -storepass
myclientTruststorePassword -alias myTrustedLDAPServerAlias
```

This command line imports a the `myLDAPServerCert.cer` certificate under alias `myTrustedLDAPServerAlias` into the `myClientTruststore.jks` keystore. The password to access the keystore is `myclientTruststorePassword`.

Remote Server API

Introduction

This section does not cover securing an instance of a TDI Server; this is discussed in "TDI Server Instance Security" on page 73. Instead, this section discusses how client applications can contact a server.

IBM Tivoli Directory Integrator supports the concept of a Remote API (also known as just "Server API"), where client tasks can invoke tasks on a remote TDI Server by means of an access layer called RMI.

Note: The "remote Server" could very well be running on the same machine as the client application, for example if you start up a Server instance on your local machine and then access it using the

Remote API through the loopback address, 127.0.01. All concepts discussed below are still valid, even though the remote Server runs locally.

The Server API calls address the following areas:

- getting Server information
- getting information for components installed on the Server
- reading and writing to configuration(s) loaded by the Server
- loading new configurations into the Server
- starting, querying and stopping AssemblyLines and EventHandlers
- cycling through AssemblyLines

The Server API itself is documented in the TDI Javadocs (<*TDI_installation_directory*/docs/api; you can launch a browser to display this documentation by selecting **Help>Low Level API** in the CE). The package of interest in this context is `com.ibm.di.api`. Also see the chapter called "Using the Server API" in *IBM Tivoli Directory Integrator 6.1.1: Reference Guide*.

The Config Editor uses the Remote API to implement the concept of a Remote Config Editor, which provides support for platforms with no native Config Editor. It can be used to read/write/execute a configuration on a remote Server. The basic idea is to provide a uniform interface for both remote and local Config files. For some aspects of the Remote Config Editor, see “Using the Remote Config Editor” on page 79.

The Server API is configured through a set of system properties. These properties are specified in the `global.properties` configuration file of the TDI Server. Some of the properties, in turn, point to additional configuration files and keystore files.

The Server API provides a number of security-related features (which both TDI Solution-based clients as well other client applications have to deal with). There are three aspects to Server API Access Security:

1. “Server API SSL remote access” on page 60 (which secures the transport channel to a remote TDI Server),
2. “Server API authentication” on page 61 (which handles the client authentication to a TDI Server),
3. “Server API Authorization” on page 68 (which handles the client authorization to a TDI Server, i.e. what the client is allowed to do once authenticated).

Configuring the Server API

The relevant properties are:

Property	Description
api.on	if set to **true** the Server API is initialized on startup and can be used; otherwise the Server API is not initialized and cannot be used. All other properties whose names start with ″api.″ are only taken into account if `api.on` is set to **true**.
api.config.folder	The configuration files placed in this folder can be edited through the Server API. At startup the TDI server scans this folder for the Solution Names of the config files located there. Also see “Optional Config instance ID in a Config file” on page 58 for more information. Configuration files placed in other folders cannot be edited through the Server API.
api.user.registry	specifies the Server User Registry file name.
api.user.registry.encryption.on	if set to **true** the Server API will decrypt the Server User Registry file on startup.
api.remote.on	if set to **true** the remote RMI part of the Server API is initialized and can be used; otherwise the remote RMI part of the Server API is not initialized and cannot be used.

Property	Description
api.remote.ssl.on	if set to **true** SSL with client and server authentication will be used on RMI connections of the Server API and its JMX layer; the Server API will use the Server certificate and private key (the one specified through the `com.ibm.di.server.keystore` and `com.ibm.di.server.key.alias` properties) for SSL connections. RMI clients need to trust that certificate. If set to **false** no SSL is used for client connections and no authentication and authorization is performed; connections are accepted from the local host and from hosts listed in the `api.remote.nonssl.hosts` property; if `api.remote.nonssl.hosts` is empty only connections from the local host are accepted.
api.remote.nonssl.hosts	specifies a list of IP addresses to accept non SSL connections from (host names are not accepted). Use space, comma or semicolon as delimiter between IP addresses. This property is only taken into account when `api.remote.ssl.on` is set to **false**.
api.remote.naming.port	the port on which the RMI registry listens for requests.
api.truststore	specifies the keystore file that contains the public certificates of all remote users of the Server API.
api.truststore.pass	the password for the keystore file specified through the `api.remote.server.truststore` property.
api.jmx.on	if set to **true** the JMX layer of the Server API is initialized on startup and can be used; otherwise the JMX layer is not initialized and cannot be used.
api.jmx.remote.on	if set to **true** the remote JMX interface (as defined by JSR160) is initialized and can be used; otherwise the remote JMX interface is not initialized and cannot be used.

Note: The Java system properties that the Server API uses for its configuration are the same, regardless of whether the client is a Java program or a different instance of the TDI Server. What should be noted though is that the way these Java system properties are set might be different. In TDI these properties are normally set by editing the global.properties/solution.properties files, whereas in a Java program they can be specified either at the command line using the -D Java command line switch or by using Java code within the Java program using the `java.lang.System.setProperty(key,value)` standard Java method.

Optional Config instance ID in a Config file

Solution Name - the Configuration Instance ID: When a configuration file is loaded by the TDI Server, it becomes a running **configuration instance**. Each configuration instance has its own configuration id. No two configuration instances running at the same time are allowed to have the same configuration id (a configuration id uniquely identifies a running configuration instance within the TDI Server).

When a configuration instance is started off a configuration file, the TDI Server first checks if the configuration file has a defined **Solution Name** (a configuration field of the Solution Interface configuration screen). If the Solution Name is present and non-emtpy, the Server uses this name as the configuration instance id. If the Solution Name is missing or empty, the TDI Server automatically generates a configuration id.

For example if a configuration file with an absolute file name “C:/IBM/TDI/configs/rs.xml” is loaded into the TDI Server and the file has a Solution Name set to “mysoluname”, then the id of the spawned configuration instance will be “mysoluname”. If the same configuration had no Solution Name defined, the configuration instance id would be something like “C__IBM_TDI_configs_rs.xml”.

Note: Clients of the Server API should not second-guess the values of generated configuration instance ID's, since the algorithm for this is subject to change in the future.

The only guarantee is that if a configuration instance once existed under some automatically generated configuration id, then certain artifacts such as tombstones and system logs can be accessed later using the same configuration id. There is no guarantee, however, that if the same configuration file is run again, the newly spawned configuration instance will have the same automatically generated id as before.

The Solution Name (if configured) must be a valid file name on the platform on which the TDI Server is currently running. The reason for this restriction is that the configuration instance id (which derives from the Solution Name) is used when storing certain configuration-instance-specific information, such as the System Logs.

If a configuration with a Solution Name which is not a valid file name is started, the TDI Server will automatically transform the Solution Name into a configuration id which is a valid file name and will log a warning.

If Solution names are used in Config files, then it is these that must appear in the User Registry instead of absolute file-system paths, for configurations which have such names. For example if a configuration file with an absolute file name "C:/IBM/TDI/configs/rs.xml" has a Solution Name set to "mysoluname" then sections of the User Registry regarding that configuration must look like this:

```
[CONFIG]:mysoluname
```

If the same configuration had no Solution Name defined, the sections about it in the User Registry would have to look like this:

```
[CONFIG]:C:/IBM/TDI/configs/rs.xml
```

Using Solution Name instead of Config file path: In TDI 6.1 and previous releases starting a config instance as well as the check-in/check-out functionality of the Server API required the URL (file path) of the config file to be provided. This is no longer necessary in TDI 6.1.1, because the same Server API interface methods can be passed the corresponding Solution Name instead. This is a user convenience as Server API clients like the AMC and CLI will now accept user-friendly Solution Names instead of cryptic config file paths.

The config file path has a higher priority than the Solution Name. This means that if the method for starting a config instance (for example) is passed a string (either a config file path or the corresponding Solution Name) and it is a valid config file path then the method will treat this value as referring to this config file. If there is a config file and a Solution Name which are identical as strings then the config file path takes precedence. This behavior ensures backward compatibility with previous versions of TDI when there were no Solution Names.

At TDI Server startup time, only Configs residing in the TDI configs folder (as specified by the `global.properties/solution.properties` file parameter **api.config.folder**) as well as those residing in the Solution Directory can be referred to by their Solution Name.

Scanning the configs folder for Solution Names: At startup the TDI Server scans the configs folder (specified by the **api.config.folder** property in `global.properties/solution.properties`) for the Solution Names of the config files located in the configs folder. The Server then builds an internal map which maps Solution Names to config file paths so that Solution Names can be used in place of config file paths.

The following rules are used when scanning the configs folder:

1. If the file name has an extension of ″.cfg″ - return the file name (these would be very old-style Configs)
2. If a config can be loaded successfully by the config driver, then check for solution name
3. If a config can **not** be loaded by the config driver,
   a. if the file name has an extension of ″.xml″ - return the file name
   b. different extension - ignore the file and don't return anything

This would lead to the following situations depending on how the TDI server is started:

Table 3.

TDI server in Secure mode	TDI server in Normal mode
PKI-encrypted config - solution name displayed (if existing)	PKI-encrypted config - file name displayed (if extension is .cfg or .xml)
unencrypted config - file name displayed (if extension is .cfg or .xml)	unencrypted config - solution name displayed (if existing)
password-encrypted config - file name displayed (if extension is .cfg or .xml)	password-encrypted config - file name displayed (if extension is .cfg or .xml)
some other file which is not a TDI config (text or binary) - file name displayed (if extension is .cfg or .xml)	some other file which is not a TDI config (text or binary) - file name displayed (if extension is .cfg or .xml)

We do, however, not recommend that you store files other than valid Config files (XML format or .cfg file format) in the configs folder. During attempts to parse any non-config file, errors may be reported and the file will be ignored – this will not affect the proper operation of the Server, however.

Remote Server API access on a Virtual Private Network

When the Remote Server API is accessed from a client on a Virtual Private Network (VPN), the VPN assigns an IP address to the Server API client machine. This VPN-assigned IP address needs to be specified in an RMI Java system property. If the Server API client is the Remote Config Editor, then this property can be set in `global.properties` and `solution.properties` by adding the following line to the properties files:

```
java.rmi.server.hostname=<IP_address>
```

Where *IP_address* is the VPN-assigned IP address.

If the Server API client is a custom Java program, then this property can be set from within the Java code in the following way:

```
java.lang.System.setProperty("java.rmi.server.hostname", "IP_Address");
```

where *IP_address* is the VPN-assigned IP address.

Please note that the RMI Java system property needs to be set before any Server API related RMI code.

Server API access options

The Server API can be used in a variety of ways:

- Access the Server API from the Remote Config Editor through a network connection
- Access the Server API from TDI components running in a remote TDI server (remote Server API access); examples of such components would be:
  - System Queue Connector
  - Server Notifications Connector
  - etc.
- Access the Server API from within the same Java Virtual Machine of the TDI Server (local Server API access); in this case the Server API can be reached from JavaScript™ in hooks or from the Script Component in addition to the options above.

Server API SSL remote access

The Server API provides two sets of interfaces – local and remote. It is only the remote interfaces that can use SSL. The local interfaces do not use SSL as the access is within the boundaries of the Java Virtual Machine. TDI can act as a server, as a client; as well as both as a client and as a server in a Server API access scenario. When SSL is used with the Server API, then a keystore and a truststore need to be configured. There are two options for configuring these. Which of them will be used depends on whether the Java System property api.client.ssl.custom.properties.on exists and on its value.

Using Server API specific SSL properties

When the Java System property api.client.ssl.custom.properties.on is set to "true", then SSL is configured through the following TDI Server API-specific Java System properties:

- **api.client.keystore** – specifies the keystore file containing the client certificate
- **api.client.keystore.pass** – specifies the password of the keystore file specified by api.client.keystore
- **api.client.key.pass** – the password of the private key stored in keystore file specified by api.client.keystore; if this property is missing, the password specified by api.client.keystore.pass is used instead.
- **api.truststore** – specifies the keystore file containing the TDI Server public certificate.
- **api.truststore.pass** – specifies the password for the keystore file specified by api.truststore.

Using the Server API specific SSL properties is convenient when your client application is using the standard Java SSL properties for configuration of another SSL channel used by the same application.

You can specify these properties as JVM arguments on the command line, for example:

```
java MyTDIServerAPIClientApp
 -Dapi.client.ssl.custom.properties.on=true
 -Dapi.truststore=C:\TDI\serverapi\testadmin.jks
 -Dapi.truststore.pass=administrator
 -Dapi.client.keystore=C:\TDI\serverapi\testadmin.jks
 -Dapi.client.keystore.pass=administrator
```

This example refers to the sample "testadmin.jks" keystore file shipped with TDI. Note that it contains both the client private key and also the public key of the TDI Server, so we use it both as a keystore and truststore.

Also these properties can be specified in `global.properties` or `solution.properties` when the client is a TDI server.

Using the standard SSL Java System properties

When the Java System property api.client.ssl.custom.properties.on is missing or when it is set to "false", then the standard JSSE system properties are used for configuring the SSL channel. Follow the standard JSSE procedure for configuring the keystore and truststore used by the client application.

You can specify these properties as JVM arguments on the command line, for example:

```
java MyTDIServerAPIClientApp
 -Djavax.net.ssl.keyStore=C:\TDI\serverapi\testadmin.jks
 -Djavax.net.ssl.keyStorePassword=administrator
 -Djavax.net.ssl.trustStore=C:\TDI\serverapi\testadmin.jks
 -Djavax.net.ssl.trustStorePassword=administrator
```

Also these properties can be specified in `global.properties` or `solution.properties` when the client is a TDI server.

Server API authentication

Server API authentication is usually referred to in the context of a remote Server API client establishing a Server API session. This scenario represents the substance of the Server API authentication logic as the Server API provides several different kinds of client authentication. But before diving into the different authentication mechanisms let us discuss the scenario in which a local client establishes a local Server API session.

Local client session

A local client session is a session established by a client which runs in the same Java Virtual Machine as the TDI server. Examples of such sessions are local sessions for access to the local Server API established from JavaScript code in hooks or in a Script component, from Connectors and Function Components which are executed as part of an AssemblyLine which runs in the same TDI server, etc. When a local client establishes a local Server API session, the client has two options:

- Do not provide a username/password pair – in this case the local Server API session is established and the client is authorized as having the 'admin' role. For more information about Server API roles please see the "Server API Authorization" section.
- Provide a username/password pair – in this case the Server API session is established only after the 'username' supplied in the username/password pair is authorized according to the Server API Authorization logic described in the "Server API Authorization" section. This option would normally be used when a specific user ID is needed for authentication – for demos, prototyping, etc.

Remote client session

A remote client session is a session established by a client which does not run in the same Java Virtual Machine as the TDI server. Examples of such sessions are sessions for access to a remote Server API established from the Config Editor, or a Java application wishing to access a TDI Server. For access of this kinf there are the following methods of authentication to the TDI Server:

SSL-based authentication

This is the only authentication mechanism available in TDI 6.0. SSL-based authentication is based on a two-stage verification of the client's credentials.

1. First the TDI server verifies that a client (represented by its SSL certificate) has the right to access the TDI server by checking whether the client's SSL certificate is contained in the TDI server's truststore, i.e. checks whether the TDI server trusts this client. Checking whether the client's certificate is contained in the server's truststore is part of the SSL handshake sequence.

   Attention: A client certificate example, corresponding to the Server certificate example in file `testserver.jks` is provided in file `serverapi/testadmin.jks`; the certificate's password is ″administrator″. As with all default security parameters you should not rely upon these and generate your own client/server certificates and specify these in the properties files. See "Certificates for the TDI Web Service Suite" on page 81.

   The truststore is kept in the file indicated by the *api.truststore* property.
2. If the truststore check is successful then the server verifies that the client SSL certificate distinguished name (DN) matches a user ID in the "Server API User Registry" on page 70. If the client certificate's DN does not match any of the user IDs in the Server API User registry file the connection request from the client is denied. This second step could be regarded as part of the authorization sequence as well.

The SSL-based authentication mechanism can be switched off in TDI 6.1.1. An additional property is available in the TDI Server configuration file `global.properties` or `solution.properties`: **api.remote.ssl.client.auth.on**. When this property is set to "true", the TDI Server will require client authentication within the SSL handshake (the TDI 6.0 mechanism for SSL-based authentication). SSL client authentication for TDI Server API does not depend on whether a username/password pair is supplied. This means that if no username/password pair is supplied, the TDI 6.0 mechanism for SSL-based authentication will be used. And if a username/password pair is supplied then the client will still need to send its SSL certificate for authentication, but the User ID for authentication (and at a later step authorization) will be taken from the username supplied.

When **api.remote.ssl.client.auth.on** is set to "false", SSL-based authentication cannot be used. When the property is not specified a value of "false" is assumed.

Username/password based authentication

This mechanism requires a client to supply a username and password on the opening of his Server API connection to the TDI server. In order to configure this authentication method an authentication hook is used.

Authentication hook: This hook allows the provision of custom JavaScript code that performs username/password based authentication. This hook allows bundlers/deployers to write customized JavaScript code, which given a username/password pair determines whether the authentication should succeed or not.

The property allowing for this custom JavaScript authentication is specified in the TDI Server configuration file `global.properties` or `solution.properties`: **api.custom.authentication**. The api.custom.authentication property points to a JavaScript text file on the disk that contains custom authentication code. If this property is not specified then the TDI 6.0 SSL-based authentication mechanism is used. When the api.custom.authentication property is specified, the JavaScript code contained in the specified file will be executed for each username/password based authentication request.

The authentication script will have access to the predefined script object userdata. This object provides the following two public members:

- **userdata.username** – will contain the name of the user requesting authentication
- **userdata.password** – will contain the password provided by the user

The script is free to perform whatever checks and authentication actions it needs. It will have to return whether the authentication is successful through the **ret** object:

- set **ret.auth = true** to specify that the authentication is successful
- set **ret.auth** = false to specify that the authentication is not successful; in this case the authentication script can provide additional information for why the authentication failed through the **ret.errordescr** attribute (for example *ret.errordescr = "Invalid user name"*) and **ret.errorcode** (for example *ret.errorcode = 1*).

The description and error code fields will be provided by the AuthenticationException thrown by the ServerAPI on unsuccessful authentication.

The authentication script will have access to the main script object. It can be used for logging custom messages in the TDI Server log file (for example main.logmsg("Authentication failed for user : " + userdata.username)).

An example authentication hook: An example authentication hook JavaScript file is available (in *TDI_installation_directory*/examples) in order to demonstrate what the JavaScript of an authentication hook could look like. This example JavaScript can also be used as the basis of real-world TDI authentication hooks. The example JavaScript demonstrates how an authentication hook can use an LDAP server (Tivoli Directory Server, Active Directory, etc.) for authenticating client requests.

The JavaScript file is named "ldap_auth.js" and is installed in the "examples/auth_ldap" TDI Server folder. To deploy this sample LDAP authentication mechanism users can copy that file to the TDI solution folder and specify api.custom.authentication=ldap_auth.js in `global.properties` or `solution.properties`. The JavaScript code in "ldap_auth.js" will try to bind to an LDAP Server with the specified username and password. If the bind operation is successful the script will indicate a successful authentication, otherwise the authentication will be rejected. The details for connecting to the LDAP Server like the server URL are specified in the "ldap_auth.js" script – this means that users will have to edit this file and set the proper connection parameters before using the script. Here is the sample "ldap_auth.js" script:

```
env = new Packages.java.util.Hashtable();
env.put("java.naming.factory.initial", "com.sun.jndi.ldap.LdapCtxFactory");
env.put("java.naming.provider.url", "ldap://192.168.113.54:389");
env.put("java.naming.security.principal", userdata.username);
env.put("java.naming.security.credentials", userdata.password);
env.put(Packages.javax.naming.Context.SECURITY_AUTHENTICATION, "simple");

main.logmsg("Authentication request for user: " +  userdata.username);

try
{
 mCtx = new Packages.javax.naming.directory.InitialDirContext(env);
 ret.auth = true;
}
catch(e)
{
 ret.auth = false;
 ret.errordescr = e.toString();
// ret.errorcode = "49";
}
```

LDAP Authentication support

The TDI Server API provides support for LDAP Authentication. This allows you to to leverage your existing LDAP infrastructures which already hold User IDs and Passwords.

LDAP Authentication Configuration: In order to use LDAP authentication the appropriate properties must be configured in `global.properties` or `solution.properties`. The list of these properties along with their descriptions follows:

api.custom.authentication
This is the same property used for username/password authentication. For more information on username/password authentication please see the "Username/password based authentication" section. This property points to a JavaScript text file on the disk that contains custom authentication code. The user may not specify this property in which case he will only be able to use the TDI 6.0 SSL-based authentication mechanism and the TDI 6.1.1 username/password authentication will not work. Set this property to "**[ldap]**" to enable the TDI 6.1.1 built-in LDAP Authentication mechanism, like this: api.custom.authentication=[ldap] All properties starting with "*api.custom.authentication.ldap.*" will only be taken into account when *api.custom.authentication* is set to *[ldap]*.

api.custom.authentication.ldap.critical
This parameter specifies the Server API behavior when the LDAP Authentication module cannot be initialized on startup. If this parameter is set to "true" the Server API initialization will fail and the Server API will not be started.

If this parameter is missing or is set to "false" the Server API will log the LDAP Authentication initialization error but the Server API will be started. An attempt to initialize the LDAP Authentication module will be made on each authentication request received by the Server API until the LDAP Authentication module is initialized.

api.custom.authentication.ldap.hostname
The LDAP Server hostname. If LDAP custom authentication is used, this is a required property.

api.custom.authentication.ldap.port
The LDAP Server port number. For example, 389 for non-SSL or 636 for SSL. If LDAP custom authentication is used, this is a required property.

api.custom.authentication.ldap.ssl
Specifies whether SSL is used to communicate with the LDAP Server. When set to "true" SSL will be used, otherwise SSL will not be used.

api.custom.authentication.ldap.searchbase
Specifies the LDAP directory location where user searches will be preformed. When this property is not specified user searches will not be performed.

api.custom.authentication.ldap.admindn
Specifies an LDAP Server administrator distinguished name that will be used for user searches. When this property is not specified anonymous bind will be used for user searches.

api.custom.authentication.ldap.adminpassword
Password for the LDAP Server administrator distinguished name.

api.custom.authentication.ldap.userattribute
Specifies the user id attribute to be used in searches. When this property is not specified user searches will not be performed. An example setting of this property would be: api.custom.authentication.ldap.userattribute=cn

If a required property is missing an exception will be thrown on initialization.

If the value of either **api.custom.authentication.ldap.searchbase** or **api.custom.authentication.ldap.userattribute** is missing no search context will be initialized and no searches will be performed during the actual user authentication. (No search means that the bind to the LDAP Server will be attempted directly with the username and password provided for authentication.)

When **api.custom.authentication.ldap.admindn** is provided a search context will be created using "simple" authentication. If an error occurs during the search context initialization, the initialization of the LDAP Authentication module will fail and an exception will be thrown.

When **api.custom.authentication.ldap.admindn** is not provided a JNDI search context will be created using JNDI "anonymous" bind.

Note: If the search context cannot be initialized using **api.custom.authentication.ldap.admindn**, authentication fails directly – no anonymous bind is attempted.

LDAP Authentication Logic: On each attempt to authenticate a user the LDAP Authentication module is passed the username and the password for the user to be authenticated. The following authentication paths are possible:

- Both **api.custom.authentication.ldap.searchbase** and **api.custom.authentication.ldap.userattribute properties** are specified :
  - If the username given for authentication ends with the value of the **api.custom.authentication.ldap.searchbase** property it is assumed that a full distinguished name is provided and no user search is performed. A bind to the LDAP Server is attempted directly with the username and password provided for authentication. If the bind succeeds the authentication is considered successful, otherwise the authentication is considered failed.
  - If the username does not end with the value of the **api.custom.authentication.ldap.searchbase** property, a search with a subtree search scope is executed against the search context created on initialization. The search query used is "(<*LDAPUserIDAttribute*>=<*username*>)" where *LDAPUserIDAttribute* is the value of the **api.custom.authentication.ldap.userattribute** property and *username* is the username given for authentication. If exactly one search result is returned, a bind to the LDAP Server will be performed with the distinguished name of the returned entry and the password provided for authentication. The authentication will succeed only if the bind to the LDAP Server is successful. In all other cases it is considered that the authentication has failed. If multiple search results are returned, authentication fails.
- At least one of **api.custom.authentication.ldap.searchbase** or **api.custom.authentication.ldap.userattribute** properties is not specified.

  In this case no searches are performed and a bind to the LDAP Server is attempted directly with the username and password provided for authentication. If the bind succeeds the authentication is considered successful, otherwise it is considered that the authentication failed.

Host based authentication

Host based authentication is used, when SSL is turned off by specifying *api.remote.ssl.on=false* in `global.properties` or `solution.properties` files. Host based authentication is configured using the *api.remote.nonssl.hosts* property. This property specifies the list of host IP addresses from which remote Server API clients can use the Server API without specifying a username/password.

The syntax of this list of hosts is: a list of IP addresses (host names are not accepted); use a space, a comma or a semicolon as a delimiter between IP addresses. An example value of this property would be:

```
api.remote.nonssl.hosts=192.168.111.222, 192.168.112.158
```

When a client using host based authentication is successfully authenticated, then the client is granted admin authorization authority. That is why adding IP addresses to this list must be done with great care. It is not advisable to use host based authentication in production environment because of its security issues. Host based authentication would normally be used while developing a solution or when doing a demo.

Summary of Server API Authentication options

The following authentication options are available:

SSL-based authentication (the mechanism available in TDI 6.0)

Only works when *api.remote.ssl.client.auth.on=true* (you will also need *api.on=true, api.remote.on=true*, *api.remote.ssl.on=true*). The user is authorized as per the rights assigned to the SSL certificate user ID in the Server API User Registry.

Note: When SSL is used and the remote client application uses Server API listener objects, then the client application must have its own certificate that is trusted by the TDI Server (this is analogous to the setup for SSL client authentication). If there is no client certificate trusted by the TDI Server, the listener objects will not work and the remote client application will not be able to receive notifications from the TDI Server.

Username/password based authentication

Only works when *api.custom.authentication* is set to a JavaScript authentication file. This authentication method works regardless of whether SSL is used and whether SSL client authentication is used. The user is authorized as per the rights assigned to the *username* user in the Server API User Registry.

LDAP authentication

This was described in section “LDAP Authentication support” on page 64, and is dependent on a number of *api.custom.authentication* settings in the `global.properties` or `solution.properties`.

Host-based authentication

Only works when *api.remote.ssl.on=false*. Then opening of Server API sessions without username/password supplied from all hosts specified by the *api.remote.nonssl.hosts* property are successfully authenticated and granted admin authority. The *api.remote.nonssl.hosts* property can be specified in the `global.properties` or `solution.properties`.

Server API JMX layer does not support username/password authentication

The remote JMX layer of the Server API does not support username/password based authentication. It ignores the *api.custom.authentication* properties. Regardless of the value of these properties and whether custom authentication is enabled or not for the Server API, the remote JMX layer performs the following authentication:

- If SSL is turned on and SSL client authentication is turned on, the remote JMX layer performs SSL-based authentication (as in TDI 6.0).
- If SSL is turned on and SSL client authentication is turned off, the remote JMX layer does not work.
- If SSL is turned off, the remote JMX client is successfully authenticated only if its host is specified on the api.remote.nonssl.hosts property, i.e. host-based authentication is assumed. In this case the client is granted admin authority.

The net result is that the Server API JMX layer does not support username/password authentication:

Server API authentication setup examples

Authentication configuration examples:

1. Non-SSL configuration & custom authentication:

   ```
   api.remote.ssl.on=false
   api.remote.nonssl.hosts=192.168.113.51, 192.168.113.52
   api.custom.authentication=ldap_auth.js
   ```

   SSL is not used.

   - Authentication requests with no username/password supplied will only succeed if they are invoked from the localhost or from 192.168.113.51 or 192.168.113.52.
   - Authentication requests with username/password supplied will only succeed if the ldap_auth.js successfully authenticates the user specified with the username and password parameters.
   - Remote JMX clients will be authenticated only when the request comes from the localhost or from 192.168.113.51 or 192.168.113.52.
2. SSL (without client authentication) & custom authentication:

   ```
   api.remote.ssl.on=true
   api.remote.ssl.client.auth.on=false
   api.custom.authentication=ldap_auth.js
   ```

   SSL is used for remote Server API communication.

   - Authentication requests with no username/password supplied will fail because neither SSL client authentication, nor host-based authentication is switched on.
   - Authentication requests with username/password supplied will only succeed if the ldap_auth.js successfully authenticates the user specified with the username and password parameters.
   - Host-based authentication is not available in this case regardless of the value of the api.remote.nonssl.hosts parameter, because api.remote.ssl.on is set to true.
   - The remote JMX layer will not be accessible. This is because SSL is turned on but SSL client authentication is not used.
3. SSL with client authentication & custom authentication:

   ```
   api.remote.ssl.on=true
   api.remote.ssl.client.auth.on=true
   api.custom.authentication=ldap_auth.js
   ```

   SSL is used for remote Server API communication and the Server requires SSL client authentication.

   - Authentication requests with no username/password supplied will succeed when the SSL certificate of the client is present in the Server's trust store (or verifiable using the certificates in the trust store).
   - Authentication requests with username/password supplied will only succeed when the SSL client authentication is successful (the SSL certificate of the client is present in the Server's trust store) and the ldap_auth.js script successfully authenticates the user specified with the username and password parameters. In this case authorization will be performed based on the username parameter from the username/password supplied and not with the user identity from the SSL client certificate.
   - Host-based authentication is not available in this case regardless of the value of the *api.remote.nonssl.hosts* parameter, because *api.remote.ssl.on is* set to true.
   - Remote JMX clients will be authenticated when the SSL certificate of the client is present in the Server's trust store (or verifiable using the certificates in the trust store).
4. SSL with client authentication & no custom authentication:

   ```
   api.remote.ssl.on=true
   api.remote.ssl.client.auth.on=true
   api.custom.authentication=
   ```

(as an alternative, the "*api.custom.authentication*" property may be missing entirely)

SSL is used for remote Server API communication and the Server requires SSL client authentication.

- Authentication requests with no username/password supplied will succeed when the SSL certificate of the client is present in the Server's trust store (or verifiable using the certificates in the trust store).
- Authentication requests with username/password supplied will not succeed because custom authentication is not configured.
- Host-based authentication is not available in this case regardless of the value of the *api.remote.nonssl.hosts* parameter, because *api.remote.ssl.on* is set to true.
- Remote JMX clients will be authenticated successfully only when the SSL certificate of the client is present in the Server's trust store.

Server API Authorization

After a client Server API session request is authenticated it needs to be authorized.

Users of the Remote API can be assigned several roles; a role defines a list of Server API calls that can be executed by the user and also defines in what context these calls can be executed. A Server API method can be executed if there is at least one role assigned to the user that allows the execution of this method in the context the user tries to execute it. For example, a role can grant the user rights to execute only specific AssemblyLines from a specific configuration. Refer to "Server API User Registry" on page 70 for details on how to create the file that holds these user rights.

Authorization is based on the user id. Depending on the authentication mechanism used the user id is retrieved in a different way:

- SSL based authentication – the user id is the distinguished name (DN) of the client's SSL certificate.
- Username/password based authentication – the user id is the username supplied in the username/password pair.
- Host based authentication – no user id can be retrieved from the client using this authentication mechanism; in this case the client session is authorized with the *admin* role.

Authorization roles

Users of the Remote API are assigned roles; a role defines a list of Server API calls that can be executed by the user and also defines in what context these calls can be invoked. For example, a role can grant the user rights to invoke only specific AssemblyLines from a specific configuration.

Several roles can be assigned to a user, including assigning the same role several times with different parameters. A Server API method can be invoked if there is at least one role assigned to the user that allows the execution of this method in the context the user tries to execute it.

There are no deny semantics – actions cannot be explicitly forbidden. The following roles apply to the Server API security model:

Read role: **read [list_of(configuration)]**	The *read* role allows the user read data from the Server's configuration(s).
	If no list of configurations is specified or the list is empy, the user is not allowed to read any configuration.
	A special value * (asterisk) can be specified for the list of configurations and this means that the user is allowed to read (through Server API calls) all configurations currently loaded by the Server.
	When the list of configurations is not null/empty and does not specify * the user is allowed to read only the configurations specified.
	The *read* role does not grant permission to start processes (AssemblyLines, EventHandlers) or apply any changes to the Server and its configurations. For example:
	`[ROLE]:read`
	`    [CONFIG]:*`

Execute role: **execute [list_of(configuration [list_of(AssemblyLines), list_of(EventHandlers)])]**	The *execute* role gives the user permissions to execute AssemblyLines and EventHandlers.
	If no list of configurations is specified or the list is empty, the user is not allowed to execute any AssemblyLine or EventHandler from any configuration.
	A special value * (asterisk) can be specified for the list of configurations and this means that the user is allowed to execute all AssemblyLines and all EventHandlers from all configurations.
	When the list of configurations is present and does not specify * the user is only allowed to start the processes from the configurations specified in the list. For each configuration specified in the list:
	• If a list of AssemblyLines is not specified the user is not allowed to execute any AssemblyLine from this configuration.
	• A special value * (asterisk) can be specified for the list of AssemblyLines and this means the user is allowed to execute all AssemblyLines from this configuration.
	• If the list of AssemblyLines is present and does not specify * the user is allowed to execute only the AssemblyLines specified in the list.
	• If a list of EventHandlers is not specified the user is not allowed to execute any EventHandler from this configuration.
	• A special value * (asterisk) can be specified for the list of EventHandlers and this means the user is allowed to execute all EventHandlers from this configuration.
	• If the list of EventHandlers is present and does not specify * the user is allowed to execute only the EventHandlers specified in the list.
	For example:
	`[ROLE]:execute`
	`    [CONFIG]:C:/TDI/rs.xml`
	`        [AL]:*`
	`        [EH]:*`
	`    [CONFIG]:C:/TDI/prototype.xml`
	`        [AL]:TestAssemblyLine`

Admin role: **admin**	The *admin* role allows the user to execute all Server API calls in every possible context. A user with *admin* role is allowed to read and modify configurations, to load new configurations, to execute AssemblyLines and EventHandlers, to read and modify server parameters. For example: `[ROLE]:admin` **Note:** Admin role is required to use the Remote Config Editor. Also see "Using the Remote Config Editor" on page 79.

The values specified in a [CONFIG] tag can be either Config file names, or Solution Names if they have been specified in the Config file. For more about Solution Names and configuration instance IDs, see "Optional Config instance ID in a Config file" on page 58.

Server API User Registry

The User Registry, identified by the *api.user.registry property* in the `global.properties` or `solution.properties` file is a text file that maintains the information about all the users of the API and their roles. This file is encrypted with the Server's certificate specified by the *com.ibm.di.server.key.alias* property from the keystore specified by the *com.ibm.di.server.keystore* property. The encryption algorithm employed is Asymmetric RSA encryption/decryption; that is why the Server certificate file must be created specifying the RSA algorithm, which is the default algorithm of the utility provided with TDI that you can use for this purpose. On startup the Server API engine decrypts and reads this file into its memory structures.

Notes:

1. The entire user registry file is encrypted as it is, block by block, in a straightforward manner using the RSA algorithm and the server public key. A digital signature or some sort of hashing is not utilized.
2. The authorization against the user registry is not optional. Currently the TDI Server has no concept of a plug-in authorization mechanism.

The contents of the Identity Registry text file is structured as follows:

```
[USER]
[ID]:<user_identifier>
[ROLE]:<role_identifier>
    [CONFIG]:<config_identifier>
         [AL]:<assembly_line_name>
         [AL]:<assembly_line_name>
         [EH]:<event_handler_name>
         ...
    [CONFIG]:<config_id>
         ...
[ROLE]:<role_identifier>
         ...
[ROLE]:<role_identifier>
         ...
[ENDUSER]

[USER]
[ID]:<user_identifier>
[ROLE]:<role_identifier>
...
[ENDUSER]
...
```

Each tag must span a single line and each tag must be on a separate line. Tabs and spaces do not matter. Empty lines may appear anywhere. The tags in the Identity Registry file and their arguments are as follows:

Tag	Argument
[USER]	No arguments; this tag serves as an opening bracket for the tags below; a **[USER]** and **[ENDUSER]** pair of tags each placed on a single line wrap the definition of a single user in the registry file. There can be multiple such pairs, each of which specify a user of the Server API.
[ID]:<*user_identifier*>	This tag is the first tag after the **[USER]** tag and its argument <*user_identifier*> is the unique identifier of the user of the Server API. This ID value is the principal's subject DN from the trust store file. The tag and the argument of the tag are placed on a single line, and there can be only one **[ID]:** tag included in a **[USER]** and **[ENDUSER]** pair.
[ROLE]:<*role_identifier*>	One of **read**, **execute** or **admin**. This tag specifies a role for the user; everything after the **[ROLE]:** tag and its argument and before another **[ROLE]:** tag or an **[ENDUSER]** tag (whichever comes first) specifies details of this user role. The tag and the argument of the tag are placed on a single line, and there can be multiple **[ROLE]:** tags included in a **[USER]** and **[ENDUSER]** pair, specifying multiple roles for that user.
[CONFIG]:<*config_id*>	Specifies the identifier of a TDI configuration, the absolute file path of the configuration. Relative file paths will not be recognized. This tag is subordinate to a **[ROLE]:** tag, and therefore the tag specifies a configuration for the role specified by that **[ROLE]:** tag. Also this tag and its argument are placed on a single line, and there can be multiple **[CONFIG]:** tags, all belonging to the superior **[ROLE]:** tag. If no **[CONFIG]:** tag is associated with a **[ROLE]:** tag this means that the list of configurations for the corresponding role definition is empty.
[AL]:<*assembly_line_name*>	Specifies the name of an AssemblyLine. This tag is subordinate to a **[CONFIG]:** tag. The tag and its argument are placed on a single line, and there can be multiple **[AL]:** tags, all belonging to the superior **[CONFIG]:** tag. If no **[AL]:** tag is associated with a **[CONFIG]:** tag this means that the list of AssemblyLines for the corresponding configuration ID is empty.
[EH]:<*eventhandler_name*>	Specifies the name of an EventHandler. This tag is subordinate to a **[CONFIG]:** tag. The tag and its argument are placed on a single line, and there can be multiple **[EH]:** tags, all belonging to the superior **[CONFIG]:** tag. If no **[EH]:** tag is associated with a **[CONFIG]:** tag this means that the list of EventHandlers for the corresponding configuration ID is empty.

The following is an example of an Identity Registry file:

```
USER]
[ID]:CN=Stan, OU=TDI, O=IBM, C=US
[ROLE]:admin
[ENDUSER]

[USER]
[ID]:CN=John, OU=TDI, O=IBM, C=US
[ROLE]:read
    [CONFIG]:*
[ROLE]:execute
    [CONFIG]:C:/TDI/rs.xml
        [AL]:*
        [EH]:*
    [CONFIG]:C:/TDI/prototype.xml
        [AL]:TestAssemblyLine
[ENDUSER]
```

```
[USER]
[ID]:CN=Peter, OU=TDI, O=IBM, C=US
[ROLE]:execute
    [CONFIG]:C:/TDI/rs.xml
        [AL]:*
        [EH]:IDSChangelog
        [EH]:ADChangelog
[ENDUSER]
```

This set of Identity Registry entries implies the following:

- This registry file specifies that user ″Stan″ is an administrator and is allowed to perform each and every Server API operation.
- John is allowed to read all configurations loaded on the Server, but can only execute processes from two configurations: from ″`rs.xml`″ he can execute all AssemblyLines and EventHandlers, from ″`prototype.xml`″ he is only allowed to execute the AssemblyLine named ″TestAssemblyLine″.
- Peter is only allowed to execute all AssemblyLines and the ″IDSChangelog″ and ″ADChangelog″ EventHandlers from the ″`rs.xml`″ configuration.

Note: The **keytool** and/or the **IKeyman** utility can be used to obtain the user ID from the trust store file. The following command line will print all users from the trust store file:

```
keytool -v -list -keystore <trust_store_file> -storepass <trust_store_pass>
```

where <trust_store_file> is the keystore file that contains the certificates of all trusted users and <trust_store_pass> is the password for this keystore file. This command line will print something like the text below for each user certificate:

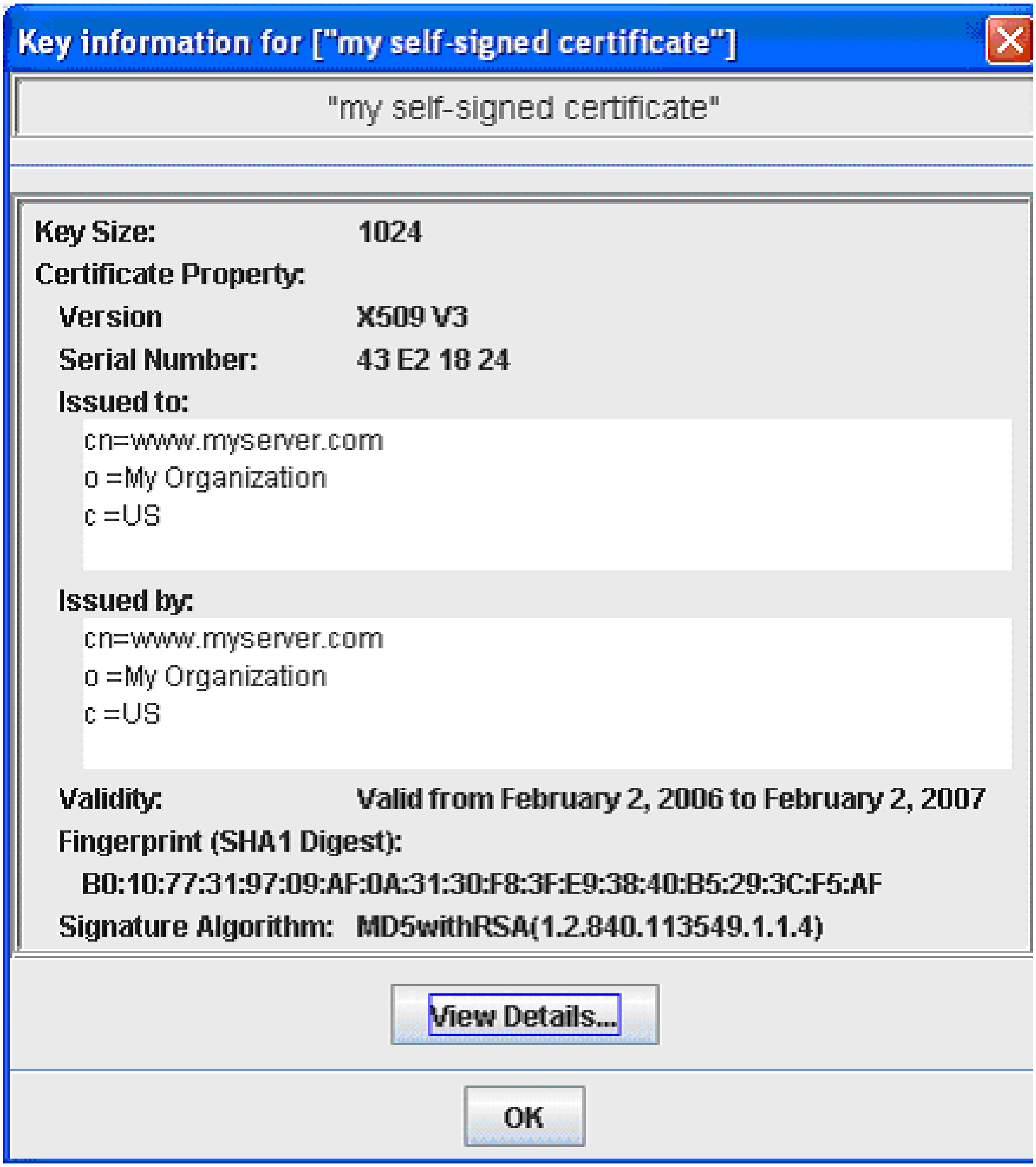
```
Owner: CN=Stan, OU=TDI, O=IBM, C=US
Issuer: CN=Stan, OU=TDI, O=IBM, C=US
Serial number: 408f6a34
Valid from: 4/28/04 11:24 AM until: 7/27/04 11:24 AM
Certificate fingerprints:
         MD5:  F6:EF:81:8B:4C:0F:10:E4:A0:16:99:AB:42:29:70:8B
         SHA1: FE:37:62:8B:42:2F:54:F8:F6:F3:FC:A1:DD:7D:2A:51:9A:85:09:02
```

The value of the **Owner** field **must** be specified as value for the [ID]: tag in the Identity Registry as is, including all white space and commas. For this example, the line with the ID tag will look like:

```
[ID]:CN=Stan, OU=TDI, O=IBM, C=US
```

An alternative way to obtain the user ID from the trust store file is to use iKeyman in the following way:

1. Start iKeyman.
2. From the **Key Database File** menu click **Open...**.
3. In the **Open** dialog box set the appropriate values and click **OK**.
4. In the **Password** dialog box enter the password for the truststore file.
5. Click on the certificate you are interested in.
6. Click the **View/Edit...** button. This will popup a dialog box, which contains information on the subject's DN (user ID).



"my self-signed certificate"

Key Size: 1024

Certificate Property:

Version X509 V3

Serial Number: 43 E2 18 24

Issued to:

cn=www.myserver.com
o =My Organization
c =US

Issued by:

cn=www.myserver.com
o =My Organization
c =US

Validity: Valid from February 2, 2006 to February 2, 2007

Fingerprint (SHA1 Digest):

B0:10:77:31:97:09:AF:0A:31:30:F8:3F:E9:38:40:B5:29:3C:F5:AF

Signature Algorithm: MD5withRSA(1.2.840.113549.1.1.4)

TDI Server Instance Security

This section does not deal with the specifics of client (TDI-based or other) access to a TDI Server, this is discussed in “Remote Server API” on page 56; instead, it focusses on the miscellaneous configuration files needed to set up a server instance.

The TDI Server requires a keystore containing both its private key and associated certificate/public key that is used for PKI encryption of Config Files, properties in Properties files, Server User registry files and other objects (using the RSA algorithm) as well as being used for SSL communication.

The system properties *com.ibm.di.server.keystore* and *com.ibm.di.server.key.alias* specify the keystore and the key alias of the Server’s certificate/key within the keystore. The password of the keystore and the password of the key itself (if different from the keystore password) are specified in the Server’s stash file.

(Access to a keystore is guarded by a password, defined at the time the keystore is created, by the person who creates the keystore, and changeable only when providing the current password. In addition, each private key in a keystore can be guarded by its own password.) For more information on the server's stash file please see the "“Stash File”" section.

Stash File

The Server stash file is named ″idisrv.sth″ (the name is not configurable) and it is loaded by the Server from the Solution Folder. The stash file contains the Server keystore password values encrypted with AES128 with a fixed key. A command line utility for creating a stash file is available in the TDI bin folder: *createstash.bat* or *createstash.sh*:

```
createstash <keyStorePassword> [<keyPassword>]
```

where *keyStorePassword* is the password of the key store file specified by the *com.ibm.di.server.keystore* system property and *<keyPassword>* is the password of the Server's private key specified by the *com.ibm.di.server.key.alias* system property. *keyPassword* is an optional parameter – if not specified it is assumed that the Server's private key password is the same as the keystore's password. The utility creates a stash file named "idisrv.sth" with the specified password(s) in the current directory.

Attention: IBM Tivoli Directory Integrator 6.1.1 comes bundled with a sample stash file, with a password of "server". For improved security, we strongly advise you to generate your own stash file using the aforementioned utility. Also, the stash file must be kept inaccessible, except for the actual TDI Server that needs it.

Server Security Modes

The TDI Server can run in two modes: **standard** and **secure**.

Standard mode
When run in standard mode the Server will not PKI encrypt configurations saved on disk, unless a specific Server API call that requests PKI encryption is invoked. When in this mode the Server is able to read both encrypted and unencrypted configurations.

Secure mode
When run in secure mode the Server will encrypt all configurations it saves on the disk using PKI encryption. In secure mode the Server will only be able to read and load encrypted configurations. When the system property *com.ibm.di.server.securemode* is set to "true", the Server will run in secure mode. (A system property for the use of the TDI Server can be set by adding it in the `global.properties` or `solution.properties` file or directly specify it on the java command line when starting the TDI server: *-Dcom.ibm.di.server.securemode=true*)

If the command line option -e is specified on the java command line when starting the Server, it will run in secure mode regardless of the value of the *com.ibm.di.server.securemode* system property.

Note: Pre-TDI 6.0 password-based encryption of configuration files is supported for backward compatibility. Password-based encryption is used when the user specifies a password when creating the configuration. Pre-TDI 6.0 password-based configuration encryption cannot be combined with the new PKI encryption. If you specify a password when the Server is run in secure mode, an error message will be displayed.

Working with encrypted TDI configuration files

Creating a PKI-encrypted TDI configuration file from scratch

There are two alternative ways to create an encrypted TDI configuration file from scratch.

Using a local server in secure mode:

1. At the command prompt, run the **ibmdisrv** command to start a local ("remote") server instance

   ```
   ibmdisrv -d -e
   ```

This will start a server on your own machine.

2. At the command prompt, run the **ibmditk** command to start the CE.
3. In the Config Editor, click **Remote->New Remote** to create a new configuration on the local TDI server running in secure mode.
4. Enter the IP address of the remote server (it runs on the local machine so enter the localhost IP address - **127.0.0.1**) and a name for the new configuration. Click **OK**.
5. Now the new remote configuration is open in the Config Editor and can be edited.
6. When you are done editing the new configuration, click **File->Save**. This will cause the local server to save the new configuration to an encrypted file.
7. Verify that the .xml file which the server just created is encrypted (the new configuration file will be created in the ″configs″ folder in the TDI working folder of the server).

Using the cryptoutils command line tool:

1. Create a normal un-encrypted TDI configuration file using the Config Editor.
2. Use the `cryptoutils` command line tool to encrypt this configuration file as described in the ""The TDI Encryption utility" on page 76" section.
3. In order to run this encrypted configuration file you need to start the TDI server in secure mode as described in the "Server Security Modes" section.
4. In order to edit this encrypted configuration file you can use one of two options described in the ""Editing an encrypted TDI configuration file"" section.

Editing an encrypted TDI configuration file

There are two alternative options for editing an encrypted TDI configuration file.

- You can first decrypt the encrypted configuration file using the `cryptoutils` command line tool as described in the ""The TDI Encryption utility" on page 76" section. Then you can edit the decrypted configuration using the Config Editor and finally you can encrypt back the modified configuration file using the `cryptoutils` tool.
- You can use the Remote Config Editor to edit an encrypted configuration on a TDI server running in secure mode.

Encrypting/decrypting an existing TDI configuration file

The `cryptoutils` command line tool must be used in order to encrypt or decrypt an existing TDI configuration file. For more information as well as examples please see the ""The TDI Encryption utility" on page 76" section.

Standard TDI encryption of global.properties or solution.properties

The `global.properties` or `solution.properties` files store a number of properties, some of which can represent sensitive data such as passwords. In order to protect this sensitive data TDI 6.1.1 is capable of encrypting this data. Here is how this works.

All properties whose names are prefixed with *{protect}*- will be PKI encrypted by the Server using the Server's public key. The Server's key is specified by the *com.ibm.di.server.key.alias* property from the keystore specified by the *com.ibm.di.server.keystore* property. For example, if you want to encrypt a property *com.ibm.di.any.property* you can add the following line in the `global.properties` or `solution.properties` file:

```
{protect}-com.ibm.di.any.property=some_value
```

The next time the Server runs it will detect that this property has to be encrypted and it will immediately overwrite the file, writing the plain text value ″some_value″ in encrypted form.

Note: On some operating systems (z/OS, Linux/UNIX systems if so configured) the file might not be accessible for writing. In this case the server outputs a warning message that the file has not been written/encrypted.

Apart from the properties for which {protect}- has been specified, the TDI Server always encrypts the following properties as well:

- `com.ibm.di.store.jdbc.password`
- `javax.net.ssl.trustStorePassword`
- `javax.net.ssl.keyStorePassword`
- `api.truststore.pass`

Whenever the Server loads the properties file and detects that any of the predefined set of properties is not encrypted, it will rewrite the file, saving all properties from the predefined set in encrypted form.

Protecting the properties in `global.properties` or `solution.properties` is also accessible from the “Global-Properties” and “Solution-Properties” Property Stores accessible from the “Properties” folder in the Config Editor GUI.

Encryption of properties in external property files

Properties stored in external property files can be protected by encryption in just the same way as properties in the `global.properties` or `solution.properties` can. For more information on encrypting properties stored in these files please see the ““Standard TDI encryption of global.properties or solution.properties” on page 75” section. The syntax of properties in an external property file is as follows:

```
[{protect}-]keyword <colon | equals> [{encr}][{java}]value
```

- The optional *{protect}-* prefix signals that the value either is or should be encrypted. When the value starts with the character sequence {encr} it means that the value is already encrypted.
- The optional *{java}* value prefix signals that the value is a serialized java object. The value must be b64-encoded For example:

  ```
  {protect}-api.truststore.pass={encr}J8AKimpEutu3Bbl0Vg55F/5d5vO2kXWcNUWnCq3vINUc6K0719z9dEk3H43Ot2iTT1dZTI6FSSVin9KsCyBLm
  ```

The TDI Encryption utility

In the *TDI_installation_directory*/serverapi directory you will find a utility (`cryptoutils`) which will enable you to decrypt and re-encrypt the Identity Registry file such that you can edit the file manually. It is used as follows:

```
cryptoutils <inputFile> <outputFile> <mode: encrypt|decrypt> <keyStore> <certificateAlias> <stashFile>
```

where

`<inputFile>`
This is the file to be encrypted or decrypted.

`<outputFile>`
This is the new file that will created with the resulting data after the encryption or decryption is done.

`<mode: encrypt|decrypt>`
Specify either ″encrypt″ or ″decrypt″ to correspondingly encrypt or decrypt the input file.

`<keyStore>`
Specifiy the Server’s keystore .

`<certificateAlias>`
Specify the alias of the Server’s certificate.

`<stashFile>`
Specify the Server’s stash file that contains the password values for the Server’s keystore.

Examples:

- This creates a stashFile:

```
cryptoutils registry.txt registry_encr.txt encrypt ..\testserver.jks server ..\idisrv.sth
```

- And the following is an example of using *cryptoutils* on configuration files:

  ```
  cryptoutils myconfig.xml mynewconfig_enc.xml encrypt_config ..\testserver.jks server ..\idisrv.sth
  ```

- This command encrypts the “myconfig.xml” configuration file and saves it as “mynewconfig_enc.xml”. Now the encrypted file can be used by a TDI Server which is running in secure mode

  ```
  cryptoutils myconfig.xml mynewconfig_enc.xml encrypt_config ..\testserver.jks server ..\idisrv.sth
  ```

- This command decrypts the “myconfig_enc.xml” configuration file (possibly created by a TDI Server, which runs in secure mode). Now the decrypted configuration “myconfig.xml” can be easily modified using the Config Editor. After modifying the configuration, it can be encrypted again, so that a TDI Server in secure mode can read it.

  ```
  cryptoutils myconfig_enc.xml myconfig.xml decrypt_config ..\testserver.jks server ..\idisrv.sth
  ```

Note: the *cryptoutils* tool can be used to encrypt/decrypt files as a whole. This makes it suitable for working with the User Registry, configuration files (see the ““Server Security Modes” on page 74” section for details how the server treats encrypted configurations) and encrypted server hooks. However, the cryptoutils tool should not be used to encrypt any of the TDI properties files. The reason is that TDI properties files are never encrypted/decrypted as a whole by the server. Only particular property values are being encrypted (see section ““Standard TDI encryption of global.properties or solution.properties” on page 75” for more information).

Miscellaneous Config File features

The “password” configuration parameter type

The configuration parameters of a TDI component in a Config can be “string”, “number”, “boolean”, etc. One of the available types is “password”. If a configuration parameter is of type password, then the Config Editor shows its value in the component configuration panel as a sequence of ‘*’ characters – both when typing in a new password, and when opening an existing configuration for editing or running.

Component Password Protection

TDI saves configuration information in an XML file which contains clear text for all configuration values. This includes sensitive information like passwords. TDI supports encryption of the entire configuration file but does not encrypt or protect sensitive information when the configuration file is saved in clear text.

TDI provides a way to better protect passwords which are needed for its various components. TDI hides the passwords in a clear text configuration and provides default security for passwords that are stored in there. In order to do this component passwords are defined (stored and retrieved) in a default property store, instead of in the configuration file. In TDI 6.1.1, a user defined property store can be any system for which there is a connector, and the default property store most likely be an external properties file. All component passwords will by default go to this default property store, instead of in the configuration file (as it is in TDI 6.0). Thus, passwords can be isolated from the configuration file unless explicitly overridden by the user (may be appropriate for initial development).

Saving passwords to configured Properties

The password protection mechanism is directly related to the configuration panels offered to the user. The configuration panels, or forms, contain descriptions of each parameter and its syntax. One type of syntax is *password* which causes the Config Editor to use a password text field for editing. Whenever the value for a password syntax component parameter is changed, the value of the password is saved in an external repository, called the *Password Store*. This external repository for passwords is configured in the *Properties* page in the configuration editor (*Password-Store*) and is specified in the configuration file for the current TDI solution. If no such property store is configured the password will be saved in clear text in the configuration file.

If a default password store is configured, a unique property name is generated the first time a protected/password parameter is saved. This key will be used as the key in the password store. The same property name is written to the configuration file as a standard property reference. When the value is later retrieved, standard property resolution takes place to retrieve the actual value from the password store.

If a Password Store is specified, a unique key is generated for the password and the password is saved encrypted in the Password Store under that key. In the configuration file, the password is referenced only by that key.

If no Password Store is specified, the password appears in plain text in the configuration file.

For example:

1. Create a new solution from the TDI Config Editor
2. From the “Properties” tab of the Config Editor insert a new Property Store called “MyProps”.
3. From the “Connector” tab of the newly created Property Store, type in “MyProps.properties” in the “Collection Path/URL” field.
4. Specify that the new Property Store will be used as the Password Property Store (in the “Password store:” combo box on the “Properties” tab of the Config Editor select “MyProps”).
5. Add a new assembly line with a FTP Client Connector.
6. Enter a password in the “Login Password” field of the FTP Client Connector.
7. Save the solution and close the Config Editor.

After the above procedure, in the configuration file of the created solution will contain lines that resemble the following:

```
<parameter name="ftpPass">@SUBSTITUTE{property.MyProps:ftpPass-38ae53e8779cfd65}</parameter>
 .........
<PasswordStore>MyProps</PasswordStore>
```

...and in the “MyProperties.properties” file there will be a line like the following:

```
{protect}-ftpPass-38ae53e8779cfd65={encr}GVJC0lA7VUiW=
```

This means that the FTP password configuration in the solution file references an encrypted property from the current Password Store - “MyProps”. The property key used is “ftpPass-38ae53e8779cfd65”.

Protecting attributes from being printed in clear text during tracing

TDI solution builders need a way to protect sensitive data, such as passwords, from being printed in clear text when tracing on the solution is needed. Therefore in TDI 6.1.1 some of the methods dealing with the Attribute class have been enhanced to say whether an attribute is protected or not. If the attribute is marked as protected and tracing is on, a fixed number of stars ‘*’ will be output instead of the actual value.

When connection parameters are found in the TaskCallBlock (TCB), the values will never be logged directly by TDI. The fact that parameters were given will be logged, but not the values themselves. If the solution needs to be debugged, those values can be dumped manually, for example using scripting.

Encryption of TDI Server Hooks

Server Hook scripts are defined and made available by creating files in the “serverhooks” subdirectory of the solution directory. Scripts that contain sensitive information should be encrypted with the Server API before adding it to the directory. Scripts can be encrypted by using the “serverapi/cryptoutils” script. Please note that the TDI server will only decrypt script files which have the “.jse” filename extension. The ″.jse″ extension indicates to the TDI server that the script file is encrypted. That is why after encrypting a Server Hook script file make sure to change its filename extension to “.jse”.

Remote Config Editor and SSL

The Config Editor used to edit remote Config Files (that is, Config files on a remote system) is called the Remote Config Editor (Remote CE). The TDI 6.1.1 Remote CE is capable of starting AssemblyLines and EventHandlers in configurations opened for editing. The Remote Config Editor is a client of the Server API of the remote TDI Server. Consequently the Remote Config Editor is authenticated and authorized as a client of the Server API. In order for this to work when SSL is used:

1. The server to which the Remote Config Editor connects must be configured to require SSL client authentication. This is a configuration of the Server API – for details see the ""SSL-based authentication" on page 62" section.
2. The Remote Config Editor TDI instance must be configured to supply SSL client authentication. This is configured in a uniform way for all SSL TDI clients.

This SSL client authentication is needed because the Remote Config Editor uses listener objects so that it can be notified when an AssemblyLine or an EventHandler has terminated and for this to work with SSL both the client must trust the server identity and server must trust the client identity.

Using the Remote Config Editor

Using a Remote Config Editor is a little different from using a local CE. When running a remote Config Editor to manage a Config on a remote system, you need to be mindful of restrictions that apply to the CE in remote mode. Notable restrictions include:

- When editing Config files locally, it is sufficient to have appropriate file system access (read and write) to the Config file. However, when editing a remote Config, you need to have Admin privilege on the remote Config Instance.
- Connections to a data source (via Connect buttons in mapping panels) are evaluated locally.

  For example: ldap://localhost:389 will result in the Configuration Editor (CE) attempting to connect to the local LDAP server, rather than to the LDAP server on the remote config file machine (i.e. the z/OS machine).
- The *tidlibrary* (Resources Tab at bottom left) is a concept always local to the CE. This means that dragging AssemblyLines, or components from AssemblyLine to tdilibrary will store the AssemblyLine locally - not in a tdilibrary on the remote machine.

  Dragging AssemblyLines from the local tdilibrary to the remote AssemblyLine is not supported.
- Similar to the tdilibrary, creating/saving packages in the CE will save them on the local file system.
- WebServices related connectors and function components generate the WSDL file, .jar files (using Complex Type Generator), etc. locally. These components are not generated on the remote system to which the CE is connected and need to manually uploaded to the remote system for deployment.
- The remote CE only allows editing and viewing of those Configs that are present in the folder specified by the api.config.folder property.
- Manage System Store, View System Store and other System Store operations (such as deleting the iterator state store key, etc) that are available in the CE, work with the local system store and not with the remote TDI machine's System Store. Only when the AssemblyLine (AL) is executed does the AL connect to the remote System Store, because at AL execution time, the AL is running inside the remote JVM.
- The Parameter Substitution editor (available with Cntl+E) will show only the local properties, and not the properties set on the remote system. Similarly, creating and saving a new property store (file type) will store the property store (file) locally.
- The Config Editor used to edit remote Config Files is subject to the Server API authentication and authorization, since the CE acts as a client application. Therefore, in order to use the CE in this way you need *admin* access on the Remote Server.
- The Remote Server itself needs to have sufficient access to the local file system where the Config Files are stored. If the Config Files are stored on a read-only file system or a file storage location where the user ID under which the Remote Server is running does not have write access to, editing remote Configs is not possible.

Web Admin Console Security

See “AMC and AM Security” on page 131.

Summary of configuration files and properties dealing with security

Table 4. The table of configuration files that were discussed above and what is contained in each.

Configuration file	Location	Description
	<TDI_home> /etc	Main Server configuration file
solution.properties	Solution folder	Copy of global.properties used by the current solution
registry.txt	*<TDI_home>*/serverapi	User registry for Server API – defined by “api.user.registry” property in global.properties

Note: registry.txt can be encrypted/decrypted using the TDI cryptoutil tool. The cryptoutil tool should not be applied on `global.properties` or `solution.properties` - individual property values can be encrypted but not the whole properties file.

Table 5. The table of properties that are referenced above, characteristics about them, what they do, what their value can be, what they are used for.

Name	Possible values	Description
com.ibm.di.server.securemode	true/false	Switch on/off secure mode
com.ibm.di.server.keystore	file name	Keystore used by server
com.ibm.di.server.key.alias	Key alias	Alias for key from keystore
api.on	true/false	Switch on/off Server API
api.user.registry	file name	Server API users registry file
api.user.registry.encryption.on	true/false	Indicates whether user registry is encrypted or not
api.remote.on	true/false	Switch on/off remote Server API
api.remote.ssl.on	true/false	Switch on/off SSL for remote Server API
api.remote.ssl.client.auth.on	true/false	Switch on/off SSL client authentication for remote Server API
api.truststore	file name	Truststore used by Server
api.truststore.pass	*	Password for truststore
api.remote.nonssl.hosts		Specifies a list of IP addresses to accept non SSL connections from.
api.custom.method.invoke.on	true/false	Specifies if the Server API methods for custom method invocation are allowed to be used.
api.custom.method.invoke.allowed.classes		Specifies the list of classes which can be directly invoked by the Server API methods for custommethod invocation
api.custom.authentication	Script file name or “[ldap]” for built in LDAP Authentication	Specifies custom authentication method
api.custom.authentication.ldap.*		Set of properties for LDAP authentication configuration

Table 5. The table of properties that are referenced above, characteristics about them, what they do, what their value can be, what they are used for. (continued)

Name	Possible values	Description
javax.net.ssl.*		Standard JSSE set of properties for keystore, truststore and their passwords

Note: All properties listed in the above table can be protected by encryption using the {protect}- prefix (see section ""Standard TDI encryption of global.properties or solution.properties" on page 75" for details).

Miscellaneous security aspects

HTTP Basic Authentication

Some TDI components give you the opportunity to use HTTP Basic Authentication as authentication mechanism. As the name says it is basic (simple) authentication. HTTP Basic Authentication should not be considered secure for any particularly rigorous definition of secure, because the credentials are base64 encoded and they can be easily decoded by someone. You should use more complex schemes to protect their data (for example a combination of turned on SSL and HTTP Basic Authentication). If the component supports HTTP Basic Authentication, then you will see the following parameter:

authenticationMethod
: Specifies the type of HTTP authentication. If the type of HTTP authentication is set to `Anonymous`, then no authentication is performed. If `HTTP basic authentication` is specified, HTTP basic authentication is used with user name and password as specified by the `username` and `password` parameters.

Lotus Domino SSL specifics

The Domino APIs for SSL do not use JSSE, and are instead Domino-specific. This means that the TDI truststore and keystore (see section "Client SSL configuration of TDI components" on page 49) do not play any part in SSL configuration for the Domino Change Detection connector. For SSL configuration of the Domino Change Detection connector, a TrustedCerts.class file is used. This file is generated every time the DIIOP process starts (in the Domino Server) and must be in the classpath of TDI (i.e. the *ibmdisrv* or *ibmditk* shell scripts which start the TDI server and TDI Config Editor respectively). The user must copy the TrustedCerts.class to a local path included in the CLASSPATH or have the Lotus\Domino\Data\Domino\Java of your Domino installation in the classpath. Whether the TDI truststore or keystore are set or not in `global.properties` (or `solution.properties`) is of no consequence to this connector.

Note: Note: The above is related to the configuration of SSL for the Notes Connector and the Domino Change Detection Connector since they use SSL over IIOP.

Certificates for the TDI Web Service Suite

The cn= portion of the distinguished name (dn) of a certificate to be used with the TDI Web Services Server Connectors must match the DNS name or IP address of the host computer on which TDI is running. Otherwise an Exception is thrown, because the client will not be able to establish an SSL connection to the TDI Web Services Server Connector. An example of the cn= portion of the distinguished name of a certificate follows: `cn=www.myserver.com`. (This constraint about the distinguished name in the server's certificate comes from the HTTPS protocol – see rfc2818 "HTTP over TLS".) Note: If TDI needs to use both a client and a server certificate only the default certificate configured in `global.properties` or `solution.properties` is used, then this must be the same certificate. An alternative would be to write a custom implementation of the SSLSocket or the SSLServerSocket Java class and make it use a certificate different from the default.

Example Server certificate creation

The following command line creates a self-signed server certificate in the keystore named "MyServerKeyStore.jks".

```
keytool -alias MyServerCertAlias -keyalg RSA -genkey -dname cn=<server_ip_address>
 -validity 365 -keystore MyServerKeyStore.jks -storepass mystorepass -keypass mykeypass
```

The alias of the created certificate is "MyServerCertAlias". The RSA algorithm is used to create the key pair. The distinguished name of the certificate is the IP of the server. The certificate is valid for 365 days (one year). The password of the keystore is "mystorepass". The password of the created private key is "mykeypass". The created certificate can then be configured for use by setting the following properties in the `global.properties` or `solution.properties` file:

```
com.ibm.di.server.key.alias=MyServerCertAlias
com.ibm.di.server.keystore=MyServerKeyStore.jks
```

MQe authentication with mini-certificates

Tivoli Directory Integrator MQe components can be deployed to take advantage of MQe Mini-Certificate authenticated access. To use these MQe features, it is necessary to download and install Websphere MQ Everyplace version 2.0.1.7 and WebSphere MQ Everyplace Server Support ES06. Use of certificate authenticated access prevents an anonymous MQe client Queue Manager and/or application submitting a change password request to the MQe Password Store Connector.

For more information on configuring MQe authentication with Mini-Certificates, please see the "Authenticated MQe Access" section in Chapter 8 "MQ Everyplace Password Store" of the *IBM Tivoli Directory Integrator 6.1.1: Password Synchronization Plug-ins Guide*.

Chapter 6. System Queue

The System Queue is a TDI JMS messaging subsystem similar to the TDI System Store. It facilitates the storing and forwarding of messages between TDI Servers and AssemblyLines. The System Queue simplifies the development of TDI solutions in which asynchronous communication is required to share work amongst multiple AssemblyLines. The System Queue can utilize either the IBM WebSphere MQ or IBM WebSphere® MQ Everyplace® as its underlying JMS messaging system, as well as any other JMS system provided the JMS Script Driver can properly address this JMS system.

Note: The System Queue Connector (see *IBM Tivoli Directory Integrator 6.1.1: Reference Guide*) does not talk directly to the System Queue, but rather uses the Server API as an intermediary.

System Queue Configuration

The System Queue is configured using the following driver-specific Java properties specified in the TDI `global.properties` or `solution.properties` file:

systemqueue.on
This parameter specifies whether the System Queue is to be started and initialized on TDI Server startup. The valid values are `true` and `false`. The default value is `false`.

systemqueue.jmsdriver.name
This parameter specifies the fully qualified name of the Java class to be used as a JMS Driver for the System Queue. This value can be the name of a user-provided class or one of the three standard TDI 6.1.1 JMS Driver implementations:

- com.ibm.di.systemqueue.driver.IBMMQe (WebSphere MQe)
- com.ibm.di.systemqueue.driver.IBMMQ (WebSphere MQ)
- com.ibm.di.systemqueue.driver.JMSScriptDriver (other JMS system by way of the JMS Script driver)

The default value is com.ibm.di.systemqueue.driver.IBMMQe.

Depending on the com.ibm.di.systemqueue.driver.IBMMQe parameter, one of the following sections is applicable:

WebSphere MQe parameters

In order to be able to use MQe as the JMS provider for the System Queue an MQe Queue Manager needs to be created. This can be done using the "MQe Configuration Utility" on page 86 bundled with TDI

systemqueue.jmsdriver.param.mqe.file.ini
This is an MQe-specific parameter that specifies the absolute file system file name of the MQe initialization file. This property is required and takes effect only if the MQe JMS driver is specified in the systemqueue.jmsdriver.name property. The default value is <*TDI_install_folder*>/MQePWStore/pwstore_server.ini. This is the default location for the MQe initialization file created by the "MQe Configuration Utility" on page 86.

WebSphere MQ parameters

These are WebSphere MQ-specific parameters; for more information about these parameters please see the MQ JMS diver initialization properties in the "System Queue Configuration Example" on page 85 section.

systemqueue.jmsdriver.param.jms.broker
(IP address and TCP port number)

systemqueue.jmsdriver.param.jms.serverChannel
(server channel defined for the MQ server instance)

systemqueue.jmsdriver.param.jms.qManager
(name of the Queue Manager defined for the MQ server instance)

systemqueue.jmsdriver.param.jms.sslCipher
(cipher suite name corresponding to the cipher selected when configuring the MQ server channel, e.g. "SSL_RSA_WITH_RC4128_MD5")

systemqueue.jmsdriver.param.jms.sslUseFlag
(true for SSL connection requested, false if not)

JMSScript Driver parameters

The JMS Driver allows you to provide connectivity to any JMS provider through scripting in JavaScript, without writing and building Java code. The JMS Driver acts as a bridge between the System Queue and a user-specified piece of JavaScript, residing on the local file system, which is responsible for creating a `javax.jms.QueueConnectionFactory` object or a `javax.jms.TopicConnectionFactory` object. These objects are obtained in a provider-specific way.

systemqueue.jmsdriver.param.js.jsfile
This is a JMS Script Driver specific parameter (i.e., taken into account when the **systemqueue.jmsdriver.name** is set to `com.ibm.di.systemqueue.driver.JMSScriptDriver`) that a specifies the name of the file that contains the user-supplied JavaScript code to handle your JMS system of choice. For more information about this parameter please see the JMS driver settings in the "System Queue Configuration Example" on page 85 section. Please note that the names of the Java properties do not have the `systemqueue.jmsdriver.param.` prefix.

systemqueue.jmsdriver.param.js.jsscript
The script body which contains JavaScript code for interfacing with the corresponding JMS provider. If this parameter is not provided, then the **systemqueue.jmsdriver.param.js.jsfile** parameter is used for loading the JavaScript to execute.

systemqueue.jmsdriver.param.user.xxxxx
These are user-defined properties which are passed by the System Queue to the configured JMS Driver implementation. For example if the following property is set:

```
systemqueue.jmsdriver.param.user.my.prop1=myvalue1
```

the configured JMS Driver will get a property with a name of `user.my.prop1` and a value of `myvalue1`.

systemqueue.auth.username
This is the user name used by the System Queue for authentication to the configured JMS system. If this parameter has not been set then the System Queue does not use authentication to the configured JMS system.

systemqueue.auth.password
This is the password used by the System Queue for authentication to the configured JMS system. This parameter is only used when the `systemqueue.auth.username` parameter has been specified.

The env JavaScript object

The piece of JavaScript executed by the JMS Driver needs access to a JavaScript object named *env*. This is an object of type java.util.Hashtable, which contains provider-specific parameters for connecting to the JMS provider. These parameters are intended to be used by the JavaScript code in order to access the specific JMS system server instance.

These parameters can be specified in global.properties/solution.properties using the "systemqueue.jmsdriver.param." prefix. For example, if a URL param is needed for some JMS system, then the following property can be set in global.properties/solution.properties:

```
systemqueue.jmsdriver.param.myjmssystem.url=myjmsserver.mydomain.com:12345
```

This definition would cause the System Queue to pass it to the JavaScript code as an entry in the env Hashtable, whose key would be “myjmssystem.url” (the System Queue removes the prefix) and whose value would be “myjmsserver.mydomain.com:12345”.

The ret JavaScript object

The piece of JavaScript executed by the JMS Driver has access to a JavaScript object named *ret*. This is an object of type com.ibm.di.systemqueue.driver.JMSScriptDriver.Ret. It is an instance of the *Ret* inner class of the JMS Script driver class. This *ret* object is to be used to return to the JMS Script driver and eventually to the System Queue the provider-specific objects which the JavaScript code obtains from the JMS system. The *ret* object can also be used to return any error information to the JMS Driver and the System Queue.

This ret object has the following members which can be set from JavaScript:

- `queueConnectionFactory` - an object of type javax.jms.QueueConnectionFactory. This is the place where the javax.jms.QueueConnectionFactory object obtained from the specific JMS system should be stored.
- `topicConnectionFactory` - an object of type javax.jms.TopicConnectionFactory. This is the place where the javax.jms.TopicConnectionFactory object obtained from the specific JMS system should be stored.
- `errorcode` - an object of type java.lang.Object. This is the place where any error information object should be stored. An example of such an object would be a java.lang.Exception object.
- `errordescr` - an object of type java.lang.String. This is the place where any textual error description should be stored.

JavaScript example for Fiorano MQ

An example configuration and JavaScript code to utilize the third-party Fiorano MQ system is provided in the <*TDI_install_directory*>/examples folder, and reproduced below:

```
var ctx = new Packages.java.util.Hashtable();
ctx.put("jms.username",   "anonymous");
ctx.put("jms.password", "anonymous");
ctx.put("jms.broker", "http://192.168.113.220:1856");
ctx.put("jms.qManager",      "fiorano.jms.runtime.naming.FioranoInitialContextFactory");

var ic = new javax.naming.InitialContext(ctx);

var queueFactory = ic.lookup("primaryQCF");
var topicFactory = ic.lookup("primaryTCF");

ret.queueConnectionFactory = queueFactory;
main.logmsg("driverFiorano.js : QueueConnectionFactory : " +  queueFactory);

ret.topicConnectionFactory = topicFactory;
main.logmsg("driverFiorano.js : TopicConnectionFactory : " +  topicFactory);
```

Note: This piece of JavaScript demonstrates how the parameters can be hard-coded in the JavaScript code. An alternative is to use the *env* JavaScript object to get any user-supplied parameters from global.properties/solution.properties. Using the *env* object for parameter retrieval would make changing the configuration easier, since only properties in global.properties/solution.properties would need to be changed, and no JavaScript code editing would be necessary. This means that users without JavaScript skills would be able to change the configuration.

System Queue Configuration Example

```
##----------------------
## System Queue settings
##----------------------
## If set to "true" the System Queue is initialized on startup and can be used;
## otherwise the System Queue is not initialized and cannot be used.
systemqueue.on=true
```

```
## Specifies the fully qualified name of the class that will be used as a JMS Driver.
# systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.JMSScriptDriver
# systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.IBMMQe
systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.IBMMQ

### MQe JMS driver initialization properties
## Specifies the location of the MQe initialization file.
## This file is used to initialize MQe on TDI server startup.
# systemqueue.jmsdriver.param.mqe.file.ini=$change$/MQePWStore/pwstore_server.ini

### MQ JMS driver initialization properties
systemqueue.jmsdriver.param.jms.broker=192.168.113.54:1414
systemqueue.jmsdriver.param.jms.serverChannel=S_s04win
systemqueue.jmsdriver.param.jms.qManager=QM_s04win
systemqueue.jmsdriver.param.jms.sslCipher=SSL_RSA_WITH_RC4128_MD5
systemqueue.jmsdriver.param.jms.sslUseFlag=true

### JMS JavaScript driver initialization properties
## Specifies the location of the script file
# systemqueue.jmsdriver.param.js.jsfile=driver.js

## This is the place to put any JMS provider specific properties needed by a JMS Driver,
## which connects to a 3rd party JMS system.
## All JMS Driver properties should begin with the 'systemqueue.jmsdriver.param.' prefix.
## All properties having this prefix are passes to the JMS Driver on initialization after
## removing the 'systemqueue.jmsdriver.param.' prefix from the property name.
# systemqueue.jmsdriver.param.user.param1=value1
# systemqueue.jmsdriver.param.user.param2=value2
# ...

## Credentials used for authenticating to the target JMS system
# {protect}-systemqueue.auth.username=<username>
# {protect}-systemqueue.auth.password=<password>
```

Security and Authentication

Encryption

Of the standard JMS Drivers, only the driver for MQ supports SSL. The MQe JMS Driver works only with a local Queue Manager – this is mandated by the MQe architecture. The JMS Script Driver is a generic driver which supports whatever the corresponding user-provided JavaScript supports.

Authentication

Some JMS systems, such as WebSphere MQ, can use or even require the use of user name and password authentication. The System Queue provides two standard properties in `global.properties` or `solution.properties` which can be used to configure and supply a user name and password to the System Queue. These properties are `systemqueue.auth.username` and `systemqueue.auth.password`. These two properties are protected by the standard TDI server encrypting of properties which are marked as `{protect}-`. In this way after these properties are set and the TDI server is started the properties' values get encrypted. For more information about these two properties please see the "System Queue Configuration" on page 83 section.

MQe Configuration Utility

The TDI 6.1.1 MQe Configuration Utility (a command line utility) creates a default MQe Queue when initially setting up the MQe Queue Manager. This default MQe Queue is named "_default". This default Queue is created for convenience only – so that a user can use the MQe Configuration Utility to set up MQe (using the appropriate MQe Configuration Utility command) and then start using the System Queue and the System Queue Connector right away.

IBM Tivoli Directory Integrator 6.1.1 provides a default configuration file, in *<TDI_install_folder>*/MQePWStore/pwstore_server.ini, the configured parameters of which will be used unless you modify them by using this utility.

Additionally the TDI 6.1.1 MQe Configuration Utility can be used to create and delete user MQe Queues to be used by the System Queue and the System Queue Connector.

Creating an MQe Queue using the MQe Configuration Utility

Typing the following command line will create an MQe Queue named “`queue_name`” using the `mqeconfig.props` configuration file:

```
mqeconfig mqeconfig.props create queue queue_name
```

Deleting an MQe Queue using the MQe Configuration Utility

Typing the following command line will delete the MQe Queue named “`queue_name`” using the `mqeconfig.props` configuration file:

```
mqeconfig mqeconfig.props delete queue queue_name
```

Authentication of MQe messages to provide MQe Queue Security

In TDI 6.1.1 access to MQe can be secured by means of authentication using the MQe Mini-Certificate Server to issue certificates to be used for authentication. For that purpose several additional properties available in TDI 6.1.1 need to be added to the `mqeconfig.props` properties file, which contains the configuration properties of the MQe Configuration Utility. For more information on these additional properties see the “Installation/Uninstallation” section.

The certificates issued by the MQe Mini-Certificate server have a configurable validity period. The default validity period is 12 months. The MQe documentation states that issued certificates should be renewed before the period expires. To enable this, the MQe configuration utility will include an option to renew certificates. Typing the following command will renew the certificates:

```
mqeconfig mqeconfig.props renewcert {client | server}
```

1. When the last command option is “client”, the following values will need to be set in the `mqeconfig.props` file:
   - **clientRootFolder** - The directory where MQe configuration instance is located.
   - **certServerReqPin** - This value is used as a one time authentication PIN for the given authenticatable entity when requesting certificate renewal from the MQe Mini-Certificate server.
   - **certServerIPAndPort** - This value is used as the destination address for MQe Mini-Certificate server requests. The format of the value is ″FastNetwork:<host>:<port>″, where host must be the machine name or TCP IP address or hostname where the MQe Mini-Certificate server is running.
   - **certRenewalEntityName** - The MQe authenticatable entity name requiring certificate renewal. Typical entity names include those below, however, any entity name configured in the MQe Mini-Certificate may be used assuming the entity does indeed exist in the queue manager registry referred to by the value of “clientRootFolder”:
     - PWStoreClient – client side MQe queue manager
     - PWStoreServer+passwords – remote queue proxy on the client side.
2. When the last command option is “server”, the following values will need to be set in the `mqeconfig.props` file:
   - **serverRootFolder** - The directory where MQe configuration instance is located.
   - **certServerReqPin** - This value is used as a one time authentication PIN for the given authenticatable entity when requesting certificate renewal from the MQe Mini-Certificate server.
   - **certServerIPAndPort** - This value is used as the destination address for MQe Mini-Certificate server requests. The format of the value is ″FastNetwork:<host>:<port>″, where host must be the machine name or TCP IP address or hostname where the MQe Mini-Certificate server is running.

- **certRenewalEntityName** - The MQe authenticatable entity name requiring certificate renewal. Typical entity names include those below, however, any entity name configured in the MQe Mini-Certificate may be used assuming the entity does indeed exist in the queue manager registry referred to by the value of “serverRootFolder”:
  - PWStoreServer – server side MQe queue manager
  - PWStoreServer+passwords – real queue on the server side.

Support for DNS names in the configuration of the MQe Queue

There is no additional coding required to support this feature. It should be noted that DNS support is really an MQe feature, since the TDI component implementations simply pass the configuration properties from `mqeconfig.props` through to the MQe APIs. The `mqeconfig.props` properties which can accept DNS name or IP address values are:

- serverIP
- certServerIPAndPort

Configuration of High Availability for MQe transport of password changes

To support high availability deployments, you have the possibility to deploy and configure multiple instances of the TDI MQe components. In some deployments, it may be necessary to configure multiple TDI MQe Password Store components. For example, if password change plugins have been configured for multiple Windows Domain Controllers—in this case, it is likely that there will separate instances of MQe client side Queue Managers with the name ″PWStoreClient″. Additionally, for each of the client Queue Managers, there will be a remote queue proxy connection to the MQe server side Queue Manager queue used by the TDI MQe Password Connector. The remote queue proxy name is ″PWStoreServer+passwords″. When you use this type of deployment scenario, the authentication certificates associated with these two MQe entities (i.e. ″PWStoreClient″, ″PWStoreServer+passwords″) will be requested and issued multiple times. This happens each time the mqeconfig utility is executed. Before executing the second and each subsequent instances of the mqeconfig utility, it will necessary to re-enable certificate issue for each of the MQe entities mentioned above.

For some deployments, you may prefer to configure the TDI MQe Password Connector such that it supports a particular high availability requirement. You may expect that an implementation supporting this type of requirement would employ multiple instances of the TDI MQe Password Connector, each with its own associated MQe Queue Manager configuration. In this case you would deploy multiple identical MQe server side configurations, allowing a network load balancer to route requests from the TDI MQe Password Store client to an available server instance. Each MQe Queue Manager on the server side will be configured using the mqeconfig utility. When this utility executes it will automatically request authentication certificates from the MQe Mini-Certificate server for the entities named ″PWStoreServer″ and ″PWStoreServer+passwords″. These represent the Queue Manager and Queue names respectively. Before executing the second and each subsequent instance of the mqeconfig utility, it will necessary to re-enable certificate issue for the two MQe entities mentioned above.

Providing remote configuration capabilities in the MQe Configuration Utility

Creating a remote MQe Queue using the MQe Configuration Utility

Typing the following command line will create a remote MQe Queue named “queue_name” using the `mqeconfig.props` configuration file:

```
mqeconfig mqeconfig.props create remotequeue queue_name targetQMname [QM_ip_or hostname comm_port]
```

In the above command line `QM_ip_or_hostname` and `comm_port` parameters are optional; if they are missing only a remote queue definition will be created. If you provide these two parameters, a Connection definition will also be created before creating the remote queue definition.

Note: A remote queue is not usable without a Connection definition. In addition several remote queues can be defined to share a single Connection. The targetQMname parameter specifies the name of the remote MQe Queue Manager.

Deleting a remote MQe Queue using the MQe Configuration Utility

Typing the following command line will delete a remote MQe Queue named "queue_name" using the `mqeconfig.props` configuration file:

```
mqeconfig mqeconfig.props delete remotequeue queue_name targetQMname
```

In the above command line the targetQMname parameter specifies the name of the remote MQe Queue Manager.

Chapter 7. System Store

IBM Tivoli Directory Integrator supports persistent storage (that is, storage of objects that survive across JVM restarts), by means of a tightly-coupled relational database, the System Store.

The product deployed by default to implement the system store is IBM DB2® for Java, also known as Cloudscape.

Third-party RDBMS as System Store

The System Store can also be configured to use other multi-user RDBMS systems, like DB2, Oracle and MS SQL*Server. This is done by specifying appropriate SQL Data Definition Language (DDL) statements and driver parameters as system properties in `global.properties` or `solution.properties`; example statements, commented out, are present in the distribution version of `global.properties` in the TDI_INSTALL/etc directory for IBM DB2.

JDBC Driver parameters provide a path to the database; additional properties are used to specify tailored SQL for certain operations TDI must be able to perform in the System Store. Multiple SQL statements can be specified per property. Each separate statement should be terminated with a semicolon. An example property could be (note that for display purposes, the statements in this document are broken up in multiple lines; however, in your property file all statements for a given property should be on one line):

```
com.ibm.di.store.create.delta.systable=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
    SEQUENCEID int, VERSION int);
    ALTER TABLE {0} ADD CONSTRAINT IDI_DS_{UNIQUE} PRIMARY KEY (ID)
```

Where {0} => will be replaced by the Table name; and

{UNIQUE} => is a special variable which can be used to generate a unique name based on the current system time.

The following are example connection parameters and statements for each of the supported RDBMS systems.

Oracle

Usage of Oracle requires that you drop the JDBC driver client library, ojdbc14.jar, in the TDI_HOME/jars directory.

JDBC connection parameters

```
com.ibm.di.store.database=jdbc:oracle:thin:@itdidev.in.ibm.com:1521:itimdb
com.ibm.di.store.jdbc.driver=oracle.jdbc.OracleDriver
com.ibm.di.store.jdbc.urlprefix=jdbc:oracle:thin:
com.ibm.di.store.jdbc.user=SYSTEM
{protect}-com.ibm.di.store.jdbc.password=password
```

Where *itimdb* is the SID of the database to be used as System Store.

Create table statements

```
com.ibm.di.store.create.delta.systable=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 SEQUENCEID int, VERSION int);ALTER TABLE {0} ADD CONSTRAINT IDI_CS_{UNIQUE} PRIMARY KEY (ID)
com.ibm.di.store.create.delta.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 SEQUENCEID int, ENTRY BLOB );ALTER TABLE {0} ADD CONSTRAINT IDI_DS_{UNIQUE} Primary Key (ID)
com.ibm.di.store.create.property.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 ENTRY BLOB );ALTER TABLE {0} ADD CONSTRAINT IDI_PS_{UNIQUE} Primary Key (ID)
com.ibm.di.store.create.checkpoint.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 ALSTATE BLOB, ENTRY BLOB, TCB BLOB );ALTER TABLE {0} ADD CONSTRAINT IDI_CR_{UNIQUE} Primary Key (ID)
```

```
com.ibm.di.store.create.sandbox.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 ENTRY BLOB )
com.ibm.di.store.create.recal.conops=CREATE TABLE {0} (METHOD varchar(VARCHAR_LENGTH), RESULT BLOB,
 ERROR BLOB)
```

MS SQL Server

Usage of MS SQL Server requires that you install a number of Microsoft client libraries in the TDI_HOME/jars directory.

JDBC connection parameters

```
com.ibm.di.store.database=jdbc:Microsoft:sqlserver://localhost:1433;DatabaseName=master;selectMethod=cursor;
com.ibm.di.store.jdbc.driver=com.microsoft.jdbc.sqlserver.SQLServerDriver
com.ibm.di.store.jdbc.user=sa
com.ibm.di.store.jdbc.password=passw0rd
```

The above connection parameters are used with these Microsoft JDBC jars:

1. Msutil.jar
2. MsBase.jar
3. MSsqlserver.jar

Note: For Microsoft SQL Server 2005, the driver jar file is `sqljdbc.jar` (only one file is required) and it can be obtained from your SQL Server 2005 installation at `<Microsoft SQL Server 2005-Install-Dir>\sqljdbc_1.1.1501.101_enu\sqljdbc_1.1\enu\sqljdbc.jar`; the driver name needs to be specified as follows:

```
com.ibm.di.store.jdbc.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
```

JDBC connection parameters (for JSQLConnect driver)

```
com.ibm.di.store.database= jdbc:JSQLConnect://itdiderver/database=reqpro
com.ibm.di.store.jdbc.driver= com.jnetdirect.jsql.JSQLDriver
com.ibm.di.store.jdbc.urlprefix= jdbc:JSQLConnect:
com.ibm.di.store.jdbc.user=administrator
{protect}-com.ibm.di.store.jdbc.password=password
```

These connection parameters are used with JSQLConnect drivers. You must download the JSQLConnect.jar file and copy it into the <TDI_HOME>/jars directory.

Create table statements

The DATA TYPE for MS SQL is IMAGE.

```
com.ibm.di.store.create.delta.systable=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 SEQUENCEID int, VERSION int);
 ALTER TABLE {0} ADD CONSTRAINT IDI_MYCONSTRAINT_{UNIQUE} PRIMARY KEY (ID)
com.ibm.di.store.create.delta.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 SEQUENCEID int, ENTRY IMAGE );
 ALTER TABLE {0} ADD CONSTRAINT IDI_DS_{UNIQUE} Primary Key (ID)
com.ibm.di.store.create.property.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 ENTRY IMAGE );
 ALTER TABLE {0} ADD CONSTRAINT IDI_PS_{UNIQUE} Primary Key (ID)
com.ibm.di.store.create.checkpoint.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 ALSTATE IMAGE, ENTRY IMAGE, TCB IMAGE );
 ALTER TABLE {0} ADD CONSTRAINT IDI_CR_{UNIQUE} Primary Key (ID)
com.ibm.di.store.create.sandbox.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 ENTRY IMAGE)
com.ibm.di.store.create.recal.conops=CREATE TABLE {0} (METHOD varchar(VARCHAR_LENGTH),
 RESULT IMAGE, ERROR IMAGE)
```

IBM DB2 for z/OS

JDBC connection parameters

```
com.ibm.di.store.database=jdbc:db2:net://localhost:50000/ididb
com.ibm.di.store.jdbc.driver=com.ibm.db2.jcc.DB2Driver
com.ibm.di.store.jdbc.urlprefix= jdbc:db2:net:
com.ibm.di.store.jdbc.user=db2admin
{protect}-com.ibm.di.store.jdbc.password=db2admin
```

Where *ididb* in the database URL is the DSN for a DB2 instance.

The above connection parameters are used with the db2jcc_license_cisuz.jar license jar file.

Create table statements

Tablespace and Indexes must be unique.

```
com.ibm.di.store.create.delta.systable=CREATE TABLESPACE TS1DSYS LOCKSIZE ROW BUFFERPOOL BP32K;
 CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, SEQUENCEID int, VERSION int) IN TS1DSYS;
 CREATE UNIQUE INDEX DSTIX1 ON {0} (ID ASC);
 ALTER TABLE {0} ADD CONSTRAINT IDI_DT_{UNIQUE} PRIMARY KEY (ID)
com.ibm.di.store.create.delta.store=CREATE TABLESPACE TS1DST LOCKSIZE ROW BUFFERPOOL BP32K;
 CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, SEQUENCEID int, ENTRY BLOB) IN TS1DST;
 CREATE UNIQUE INDEX DSIX1 ON {0} (ID ASC);
 ALTER TABLE {0} ADD CONSTRAINT IDI_DS_{UNIQUE} Primary Key (ID);
 CREATE LOB TABLESPACE DSENT11 BUFFERPOOL BP32K LOCKSIZE LOB;
 CREATE AUX TABLE TBDSEN1 IN DSENT11 STORES {0} COLUMN ENTRY;
 CREATE INDEX IXEN1 ON TBDSEN1
com.ibm.di.store.create.property.store=CREATE TABLESPACE PS3DST LOCKSIZE ROW BUFFERPOOL BP32K;
 CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ENTRY BLOB) IN PS3DST;
 CREATE UNIQUE INDEX PSIX3 ON {0} (ID ASC);
 ALTER TABLE {0} ADD CONSTRAINT IDI_PS_{UNIQUE} Primary Key (ID);
 CREATE LOB TABLESPACE PSENT31 BUFFERPOOL BP32K LOCKSIZE LOB;
 CREATE AUX TABLE TBPSEN3 IN PSENT31 STORES {0} COLUMN ENTRY;
 CREATE INDEX PSIXEN3 ON TBPSEN3
com.ibm.di.store.create.checkpoint.store=CREATE TABLESPACE TSPACE1 LOCKSIZE ROW BUFFERPOOL BP32K;
 CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ALSTATE BLOB, ENTRY BLOB, TCB BLOB) IN TSPACE1;
 CREATE UNIQUE INDEX INDX1 ON {0} (ID ASC);ALTER TABLE {0} ADD CONSTRAINT IDI_CR_{UNIQUE} Primary Key (ID);
 CREATE LOB TABLESPACE LOBSP11 BUFFERPOOL BP32K LOCKSIZE LOB;
 CREATE AUX TABLE ALSTATE1 IN LOBSP11 STORES {0} COLUMN ALSTATE;
 CREATE INDEX IXALST1 ON ALSTATE1;
 CREATE LOB TABLESPACE LOBSP12 BUFFERPOOL BP32K LOCKSIZE LOB;
 CREATE AUX TABLE ENTRY1 IN LOBSP12 STORES {0} COLUMN ENTRY;
 CREATE INDEX IXENT1 ON ENTRY1;
 CREATE LOB TABLESPACE LOBSP13 BUFFERPOOL BP32K LOCKSIZE LOB;
 CREATE AUX TABLE TCB1 IN LOBSP13 STORES {0} COLUMN TCB;
 CREATE INDEX IXTCB1 ON TCB1
com.ibm.di.store.create.sandbox.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ENTRY BLOB)
com.ibm.di.store.create.recal.conops=CREATE TABLESPACE IM{UNIQUE} LOCKSIZE ROW BUFFERPOOL BP32K;
 CREATE TABLE {0} (METHOD VARCHAR(VARCHAR_LENGTH), RESULT BLOB, ERROR BLOB) IN IM{UNIQUE};
 CREATE LOB TABLESPACE LB{UNIQUE} BUFFERPOOL BP32K LOCKSIZE LOB;
 CREATE AUX TABLE AT{UNIQUE} IN LB{UNIQUE} STORES {0} COLUMN RESULT;
 CREATE INDEX IX{UNIQUE} ON AT{UNIQUE};
 CREATE LOB TABLESPACE LS{UNIQUE} BUFFERPOOL BP32K LOCKSIZE LOB;
 CREATE AUX TABLE AE{UNIQUE} IN LS{UNIQUE} STORES {0} COLUMN ERROR;
 CREATE INDEX IN{UNIQUE} ON AE{UNIQUE}
```

DB2 for other OS

JDBC connection parameters

```
com.ibm.di.store.database=jdbc:db2:net://localhost:50000/ididb
com.ibm.di.store.jdbc.driver=com.ibm.db2.jcc.DB2Driver
com.ibm.di.store.jdbc.urlprefix= jdbc:db2:net:
com.ibm.di.store.jdbc.user=db2admin
{protect}-com.ibm.di.store.jdbc.password=db2admin
```

Where *ididb* in the database URL is the DSN for a DB2 instance.

Create table statements

```
com.ibm.di.store.create.delta.systable=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 SEQUENCEID int, VERSION int);
 ALTER TABLE {0} ADD CONSTRAINT IDI_MYCONSTRAINT_{UNIQUE} PRIMARY KEY (ID)
com.ibm.di.store.create.delta.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 SEQUENCEID int, ENTRY BLOB );
 ALTER TABLE {0} ADD CONSTRAINT IDI_DS_{UNIQUE} Primary Key (ID)
com.ibm.di.store.create.property.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 ENTRY BLOB ) ;ALTER TABLE {0} ADD CONSTRAINT IDI_PS_{UNIQUE} Primary Key (ID)
com.ibm.di.store.create.checkpoint.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 ALSTATE BLOB, ENTRY BLOB, TCB BLOB );
 ALTER TABLE {0} ADD CONSTRAINT IDI_CR_{UNIQUE} Primary Key (ID)
com.ibm.di.store.create.sandbox.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
 ENTRY BLOB )
```

The remainder of this chapter will discuss the operational aspects of using Cloudscape, in particular in conjunction with using Cloudscape to hold your System Store.

Note: With regards to 3rd party RDMSs, in order to hold encrypted password values you may need to dimension the fields that hold them quite large. A typical small password might use as much as 178 characters. It depends on both your server's key, and the length of the unencrypted data you try to store (in bytes). Since this is a blocked encoding a larger password might use the same space, or double or triple that amount. Also, the size of the block depends on the server's key. One way to find the size you need, is to store the password (protected) to a file first, and then look at that file to see how many characters were used.

Cloudscape can run in either of two modes: **embedded** and **networked**. By default, as specified in the `global.properties` file, Cloudscape is configured to run in embedded mode, and as such runs as a separate thread within the JVM when required. Startup and shutdown of Cloudscape are automatic in embedded mode. However, when run this way, this Cloudscape thread claims exclusive access to the database files. This can become problematic when different JVMs, each with their own Cloudscape thread, try to access the same System Store.

The following causes a new, independent JVM to be started, triggering an access conflict when more than one JVM is active at any given time:

- A command line invocation of the IBM Tivoli Directory Integrator Server with a config file, causing one or more AssemblyLines to run or one or more EventHandlers to be activated
- Startup of the Config Editor (GUI)
- Startup of an AssemblyLine or EventHandler from within the Config Editor

None of these actions by themselves will cause the Cloudscape thread to start. However, the Cloudscape thread does start if access to any of the objects in the System Store is required (for example, Objects supported by the System Store such as Checkpoint/Restart info, Delta Tables and the User Property Store).

The solution to the access conflicts as outlined previously is to run Cloudscape in networked mode, which enables concurrent access to the System Store.

Also, when configured in networked mode, you can work with multiple instances of Cloudscape databases booted as System Stores. However, you will have to ensure that the Cloudscape server is started before using it. You can also configure a Cloudscape instance to work with a specific Configuration file instance.

Configuring Cloudscape Instances

To configure and manage multiple Cloudscape instances and to provide facilities to start, stop and restart Cloudscape servers in networked mode a menu option called **Store** is provided in the TDI Config Editor. Many of the configuration options listed here take default values from the `global.properties` file, which was the configuration base for previous versions of TDI; now, additional System Store parameters are contained in the `CSServerInfo.xml` file.

The **Store** menu option also provides ways to configure the System Store to use IBM DB2 as the backend RDBMS. It has the following options:

- "Manage System Stores"
- "View System Store" on page 97
- "Network Server Settings" on page 97

Manage System Stores

The Manage System Stores panel provides the options to configure and manage multiple Cloudscape network server instances. It also provides options to configure and use other RDBMs as the System Store by exposing the create table syntax for the internal tables used by the System Store.

Servers
The Servers group box will be enabled only if Cloudscape is configured to run in networked mode. If Cloudscape is configured in embedded mode or if another RDBMS like IBM DB2 is configured as System Store then this group is disabled.

Load SYSIBM
If this check box is selected then the Cloudscape server checks whether the SYSIBM schema exists while booting each database and loads the schema if it does not exist. You should select this option if you want to access databases created by an embedded Cloudscape server.

Hostname
Specify the hostname or the IP address of the Cloudscape network server. This drop down also will have the list of all the servers that you would have configured or used before. The default hostname is *localhost*.

Port The port number of the Cloudscape networked server. The default port number is *1527*.

Start Start an instance of the Cloudscape network server on the specified port. The default port 1527 is used if the port is not specified explicitly.

Note: You can start the network server only on your local machine.

Stop Stop an instance of the Cloudscape networked server running on the specified **Hostname** and **Port**. The default hostname and port numbers are used if you do not specify these values.

Restart
Restart a Cloudscape networked server running on the local machine.

Note: Restart functionality, like the start server functionality, will work only on the local machine.

Start Mode
This specifies the start mode of the Cloudscape networked server. If the mode is set to manual, then you will have to start the server manually. If the mode is set to automatic then the Cloudscape networked server will automatically be started when the CE is launched or when you run the `ibmdisrv` command from the command line.

Connection Properties
This group contains the properties needed to make a connection to the specified database.

Database
Specify the database name or the URL to make a connection to. The value of the

com.ibm.di.store.database property set in the global.properties file is shown by default. If the Cloudscape server is configured to run in networked mode then by selecting the hostname and the port number you will be able to get a list of databases to specific to that server. The list of databases will be for the selected Cloudscape server instance.

Open Open and view the selected database. The selected database can be viewed in a separate tab and multiple databases can be viewed at the same time.

URL Prefix
Specify the JDBC URL prefix. The value of the com.ibm.di.store.jdbc.urlprefix property set in the global.properties file is pre-filled by default.

Username
The JDBC user name to connect to the specified database.

Password
The JDBC password to connect to the specified database. The password field is masked.

JDBC Driver
The JDBC driver class name. If this is set to com.ibm.db2.jcc.DB2Driver then the elements under the Server group will be enabled.

Note: See the JDBC Connector section in the *IBM Tivoli Directory Integrator 6.1.1: Reference Guide* for more information about this parameter.

Create table statements
You can specify the syntax of the system tables the System Store needs. These values can also be set in the global.properties files. You need to enter the syntax of the tables corresponding to the RDBMS that you choose as the System Store.

Systable
The create table statement for the IDI_SYSTABLE. By default the value of the com.ibm.di.store.create.delta.systable property set in the global.properties file is displayed.

Delta table
The create table statement for the Delta Store tables. By default the value of the com.ibm.di.store.create.delta.store property set in the global.properties file is displayed.

Checkpoint Restart
The create table statement for the Checkpoint/Restart tables. By default the value of the com.ibm.di.store.create.checkpoint.store property set in the global.properties file is displayed.

Property Store
The create table statement for the Property Store tables. By default the value of the com.ibm.di.store.create.property.store property set in the global.properties file is displayed.

Sandbox
The create table statement for the Sandbox Store tables. By default the value of the com.ibm.di.store.create.sandbox.store property set in the global.properties file is displayed.

Add Server
The information regarding Cloudscape networked server is stored in the CSServerInfo.xml file. This file can be found under the TDI installation directory. The Add server buttons adds the current configuration of the server and the databases to the CSServerInfo.xml file.

Delete Server
Delete the currently selected Cloudscape networked server from the CSServerInfo.xml file. If the selected Cloudscape network server has more than one database configured than only the selected database will be deleted from the CSServerInfo.xml file. If the selected Cloudscape networked server has just one database then the server information (hostname and port) will be deleted along with the selected database.

Set This sets the create table statements. The value is set in the system properties.

View System Store

The View System Store panel provides the means to open and view the default System Store.

Database
The value of the `com.ibm.di.store.database` property set in the `global.properties` file is shown by default.

Open When the Open button is clicked, the System Store Browser displays three items, one for each type of persistent store (Delta, CPR and Property Store). You can use this window to examine the contents of the System Store.

Close Close the currently opened database.

Delete Table
Delete the selected table in the tree.

Network Server Settings

This panel allows you to set the properties of the Cloudscape network server. To set the properties on a Cloudscape networked server the server should be running. The panel also provides a way to check if the server is already running on the specified hostname and port. You can also view the currently set properties in the Cloudscape networked server. You can also set these properties in the `derby.properties` file. This file can be found under the TDI installation directory.

Servers
Groups the network servers already configured in the `CSServerInfo.xml` file.

Hostname
Specify the hostname or the IP address of the Cloudscape network server. This drop down also will have the list of all the servers that you would have configured or used before. The default hostname is *localhost*.

Port The port number of the Cloudscape networked server. The default port number is *1527*.

Test Connection
Check if the network server is already running on the specified hostname and the port. To set the properties of the network server the server has to be running.

Start Start an instance of the Cloudscape network server on the specified port. The default port 1527 is used if the port is not specified explicitly.

Note: You can start the network server only on your local machine using this mechanism. The reason that you can specify the **Hostname** is because if the Cloudscape Server is started with the value "localhost" then it disallows any remote machine to connect to this Cloudscape server. Conversely, if the Cloudscape Server is started with the actual, real address of the local machine in the **Hostname** parameter, then remote connections will be allowed to this Cloudscape Server.

Stop Stop an instance of the Cloudscape networked server running on the specified **Hostname** and the **Port**. The default hostname and port numbers (that is, *localhost* and *1527*) are used if you do not specify these values.

Restart
Restart a Cloudscape networked server running on the local machine.

Note: Restart functionality, like the start server functionality, will work only on the local machine.

System Directory
You define the system directory when Cloudscape starts up by specifying a Java system property

called derby.system.home. If you do not specify the system directory when starting up Cloudscape, the current directory becomes the system directory.

Log Connections
This indicates whether to log the connections and disconnections. This corresponds to setting the derby.drda.logConnections property in the derby.properties file.

Max Threads
The maximum number of threads used for connections. A value of 0 indicates that there are no limits on the number of threads for connections. 0 is set as the default value. The maximum value that you can set is *2147483647* (maximum integer). However the practical maximum value that you can set is limited by the system configuration. The corresponding property in the derby.properties file is derby.drda.maxthreads.

Time Slice
Specify the time that a connection thread should work on one session's request. A value of 0 milliseconds indicates that the thread will not give up working on the session until the session ends. The maximum number of milliseconds that can be specified is *2147483647* (maximum integer).

Trace All
If this is set, then tracing will be turned on for all sessions. The corresponding property in the derby.properties file is derby.drda.traceAll = [true|false].

Trace Directory
Indicates the location of the trace files. If this value is not specified then the derby.system.home property's value will be used as the location for the trace files. The corresponding value in the derby.properties is derby.drda.traceDirectory.

Sysinfo
You can get information about the Network Server such as version and current property values, Java information, and Cloudscape database server information by using the **Sysinfo** button.

Set Set the specified properties on the Cloudscape network server. This does not set the values in the derby.propeties file but sets the network server properties directly on a running instance of the Cloudscape networked server.

Backing up Cloudscape databases

Another matter that needs to be given some thought is **backup** of the data contained in a Cloudscape database. The recommended (and simplest) way of doing this is to

- Shutdown the Cloudscape database (if running in embedded mode, shut down all TDI instances and Config Editor instances)
- Copy the entire Cloudscape directory in your TDI home directory (or whatever Cloudscape directory your global.properties file or CSServerInfo.xml file is pointing to) to a different location, and ensure that this data is safe
- Restart the Cloudscape database (if running in networked mode).

To restore a database, reverse source and destination of the copy operation in the above list of steps.

Troubleshooting Cloudscape issues

This section does not attempt to be a comprehensive Troubleshooting Guide for Cloudscape, but there are a number of symptoms that are observed sometimes in the context of usage of Cloudscape as the underlying database in TDI. These are:

Schema 'SYSIBM' does not exist error

Question:

I'm trying to use Cloudscape in networked mode and having issues. I've figured out how to start it up and I'm able to query it with sysinfo and testconnection, but when I run TDI and try to open the system store I get an error stating:

```
[com.ibm.db2.jcc.a.SQLException: Schema 'SYSIBM' does not exist]
```

How do I fix this?

Explanation:

The reason you get this error is because you are trying to boot a database that was created in embedded mode into a networked mode server without starting the server using the -ld flag. Please note that for a networked mode Cloudscape server to open an embedded mode database, the SYSIBM schema MUST be loaded. The SYSIBM schema is a special schema loaded by the Cloudscape server. The SYSIBM contains stored prepared statements that return result sets to determine metadata information.

Corrective action:

To solve this problem start the Cloudscape networked server with the ″-ld″ flag, like:

```
./dbserver start -p 1527 -ld
```

Another Instance of Cloudscape may already be booted

You may get the following error sometimes, especially when using Cloudscape in embedded mode:

```
[ERROR XSDB6: Another instance of Cloudscape may have already booted the database D:\tdi60\Cloudscape.]
```

Explanation:

Cloudscape will try to prevent two instances of Cloudscape from booting the same database (in this case `D:\tdi60\Cloudscape`). This can happen if you are running two AssemblyLines which are trying to update the same Cloudscape database running in embedded mode. This error might also crop up if you have an unclosed connection to the database.

Corrective Action:

1. If you want two AssemblyLines to update the same Cloudscape database, then the correct mode of Cloudscape should be networked mode. Please configure the Cloudscape server to run in networked mode since this mode of operation does not have that limitation.
2. You can work around this by closing the database using the **Store>View System Store** and then clicking on the close button. Even if the database is not open, just opening and closing again through the **Store>View System Store** option will help solve this problem.

Future versions of TDI will attempt to handle this situation automatically, and stop and start Cloudscape as required.

Can I use DB2 as a system store?

In TDI it is possible to use DB2 as a system store, instead of the bundled Cloudscape database system. However, some modification of system properties files will be required for this to function correctly. You will need to replace the section on Cloudscape networked mode with a section similar to the following (insert the correct parameters for your installation.

If you look at the default `global.properties` file, there are some CREATE_TABLE statements for using and setting up the system store. If you use the right syntax, you can use non-Cloudscape databases as system store. DB2, Microsoft SQL Server 2000 and Oracle 9i have been tested, but no other RDBMS has for the time being. Here is the DB2 syntax:

```
## Location of the DB2 database (networked mode)
com.ibm.di.store.database=jdbc:db2://168.199.48.4:3700/tdidb
com.ibm.di.store.jdbc.driver=com.ibm.db2.jcc.DB2Driver
com.ibm.di.store.jdbc.urlprefix=jdbc:db2:
com.ibm.di.store.jdbc.user=db2inst1
com.ibm.di.store.jdbc.password=******
com.ibm.di.store.start.mode=automatic
com.ibm.di.store.port=3700
com.ibm.di.store.sysibm=true

# the varchar(length) for the ID columns used in system store and PES Connector tables
com.ibm.di.store.varchar.length=512

# create statements for DB2 system store tables
com.ibm.di.store.create.delta.systable=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH)
 NOT NULL, SEQUENCEID int, VERSION int)
com.ibm.di.store.create.delta.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH)
 NOT NULL, SEQUENCEID int, ENTRY BLOB )
com.ibm.di.store.create.property.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH)
  NOT NULL, ENTRY BLOB )
com.ibm.di.store.create.checkpoint.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH)
 NOT NULL, ALSTATE BLOB, ENTRY BLOB, TCB BLOB )
com.ibm.di.store.create.sandbox.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH)
 NOT NULL, ENTRY BLOB )
```

Note: Each com.ibm.di.store.create.xxx statement must be specified on one line, even though they are broken up in this example for illustration purposes.

Why can't remote connections be made to my Cloudscape network server?

This may be because the Cloudscape Server has been started by passing ″localhost″ as the hostname. This disallows any remote connections to be made to Cloudscape. Stop the Cloudscape server and start it with hostname parameter specified as the machine's IP address. This can be done by going to the Config Editor's **Store -> Network** Server Settings panel.

For more details, check http://publib.boulder.ibm.com/infocenter/cldscp10/topic/com.ibm.cloudscape.doc/hubprnt22.htm

Pre-6.0 (properties file) configuration of Cloudscape

Previous versions of TDI configured the System Store using Cloudscape by means of a set of properties in the global.properties file, and version 6.1.1 still derives its base configuration from there. You should migrate any non-standard installations of Cloudscape configuration to the methods described in the previous chapter, "Configuring Cloudscape Instances" on page 95.

In the global.properties file in the installation directory, there are two sections that are concerned with the configuration of the System Store:

- The first section (enabled by default) deals with running Cloudscape in embedded (dedicated), non-shared mode.
- The second section (commented out by default) deals with Cloudscape in networked or shared mode. If you determine that you must use networked mode, comment the first section and uncomment the second.

Note that the section with configuration parameters for embedded mode specifies a relative path for the Cloudscape database; this typically means the database will be created in the Solutions directory. The set of parameters for Networked mode uses an absolute path: it points to the installation directory. If you switch between the different modes you need to be aware of this, and possibly change the path to an appropriate value, for the mode you intend to use.

Best practice is to keep databases, keystores and all other user data out of the program install directory.

When deploying networked or shared mode, startup of the Cloudscape database thread is no longer automatic. You must start a Cloudscape instance before you start your first AssemblyLine or EventHandler, and shut down the instance when you are finished with your last AssemblyLine and the last EventHandler has shut down.

To make working with the Cloudscape database more convenient, consider creating a script (″dbserver″) with the following line (this example is for Unix/Linux):

```
export DB_JAR_DIR=jars/3rdparty/IBM
export DB_CLASSPATH=$DB_JAR_DIR/derby.jar:$DB_JAR_DIR/derbyclient.jar:\
$DB_JAR_DIR/derbynet.jar:$DB_JAR_DIR/derbytools.jar
java -classpath $DB_CLASSPATH org.apache.derby.drda.NetworkServerControl "$@"
```

You may need to join the last two lines together at the "\" point.

The equivalent dbserver.bat file for Windows would be:

```
set DB_JAR_DIR=jars/3rdparty/IBM
set DB_CLASSPATH=%DB_JAR_DIR%\derby.jar;%DB_JAR_DIR%\derbyclient.jar;\
%DB_JAR_DIR%\derbynet.jar;%DB_JAR_DIR%\derbytools.jar;
java -classpath %DB_CLASSPATH% org.apache.derby.drda.NetworkServerControl %*
```

Note: The script must be started from within the IBM Tivoli Directory Integrator installation path as the working directory, as the following classpath is relative to this directory.

The following is an example of usage of this utility script:

```
Show all available commands:  ./dbserver

Start DBServer  ./dbserver start -p 1527

Start Database instance   ./dbserver dbstart /home/stadheim/DI52/Cloudscape

Stop Database instance  ./dbserver dbshutdown /home/stadheim/DI52/Cloudscape

Stop DBServer  ./dbserver shutdown
```

The full list of sub-commands that you can specify to the dbserver script, and which are sent to Cloudscape is:

- start -p *portnumber* [-ld]
- shutdown [-h *host*][-p *portnumber*]
- dbstart *databaseDirectory* [-b *bootPassword*][-ld][-ea encryptionAlgorithm] [-ep encryptionProvider] [-u *user password*][-h *host*][-p *portnumber*]
- dbshutdown *databaseDirectory* [-h *host*][-p *portnumber*]
- testconnection [-d *databaseDirectory*] [-u *user password*] [-h *host*] [-p *portnumber*]
- sysinfo [-h *host*][-p *portnumber*]
- conpool *min max* [-d *databaseDirectory*][-h *host*][-p *portnumber*]
- logconnections {on|off}[-h *host*][-p *portnumber*]
- maxthreads *max*[-h *host*][-p *portnumber*]
- timeslice *milliseconds*[-h *host*][-p *portnumber*]
- trace {on|off} [-s *session id*][-h *host*][-p *portnumber*]
- tracedirectory *traceDirectory*[-h *host*][-p *portnumber*]

When running in networked mode, the Cloudscape database is of course reachable over the network, not only by IBM Tivoli Directory Integrator instances but also by other applications using the appropriate drivers. The credentials required for such access are defined in the global.properties file, and might need to be tailored for your particular site needs. Pay particular attention to the username and password parameters as these govern integrity and security of the data.

If you often alternate between running Cloudscape in dedicated mode and in networked mode, consider having two different ″prototype″ global.properties files on your file system, one each with the correct set of parameters for each of the two modes. Just before starting a server instance, copy in place the appropriate global.properties file, according to your needs. Alternatively, use separate Solution Directories; in a Solution Directory you can have a file called `solution.properties`, which property values defined in there override the ones defined system-wide in `global.properties`.

See also

The official Cloudscape home at http://www-3.ibm.com/software/data/cloudscape, documentation at http://www-306.ibm.com/software/data/cloudscape/pubs/collateral.html.

Chapter 8. Command Line Interface (CLI)

Command Line Interface – tdisrvctl utility

The Command Line Interface (CLI) to TDI, called the *tdisrvctl* utility, is designed for remotely managing Configs, AssemblyLines, etc. This utility connects to a remote TDI server using the TDI Remote Server API, and performs the requested operations. As it is a client application interfacing to a Remote Server, it is subject to the same connection, authentication and authorization issues described in Chapter 5, "Security and TDI," on page 47.

Note: This chapter does not describe the syntax and options of the ibmdisrv command, the command to start up a TDI server. This information can be found in the *IBM Tivoli Directory Integrator 6.1.1: Users Guide*.

It exposes various command line options for the following functions:

1. Start/stop/reload TDI Configs.
2. Start/Stop Assembly Lines/Event Handlers in a particular config.
3. Display a list of configs loaded on the server.
4. Shutdown server.
5. Display config report.
6. Manage config properties through TDI-p, the TDI properties framework
7. Send custom notification events.
8. View exposed AL Operations.
9. View tombstones for terminated Configs, AssemblyLines or Event Handlers.
10. View TDI Server details.

Notes:

1. The command line utility is shipped in the <TDI_Install_Dir>/bin folder.
2. For any remote server API client (including the CLI) the property **api.remote.on** should be set to **true** and the client's IP address must be mentioned in the property **api.remote.nonssl.hosts** in the `global.properties` (or `solution.properties`) file of the remote TDI Server (if non-SSL mode is being used).
3. The remote TDI server must be running (**ibmdisrv –d**)

Command Line reference

The command has the following usage:

```
tdisrvctl [general_options] –op operation [operation_specific_options]
```

where **general_options** can be:

-h host	remote server IP address or hostname (default is localhost)
-K keystore	name of the SSL key database file
-p port	port number (default is 1099)
-P key_pwd	key file password
-T truststore	name of the SSL trust store database file
-u userID	username (for custom authentication)
-v	run in verbose mode
-w user_pwd	user password (for custom authentication)
-W trust_pwd	trust file password

-?	display command usage

And **operation** can be:

event	send custom notification events
prop	manage Config properties
queryop	query for AL operations
reload	reload running Configs
report	generate Config report or list Configs on remote server
shutdown	shutdown the server
srvinfo	view TDI server information
status	view status of Configs or ALs or EHs
start	start specific Config or ALs or EHs
stop	stop specific Config or ALs or EHs
tombstone	view tombstone entries for specific Config or AL or EH

You can display help for any particular option like this:

```
tdisrvctl –op operation -?
```

Operations

event This option can be used to send custom notification events to a particular server. All listeners registered for the particular event will receive this notification. This will allow TDI administrators to trigger listener applications based on planned custom events.

The usage for the event operation is:

tdisrvctl [general_options] **–op event –e** event_name [**-s** source] [**-d** data]

where:

-e event_name	the name of the event to send
-s source	the name of the source invoking the event (default “tdisrvctl”)
-d data	the data to be passed to an event listener (default is null)

Example:

To send an event “user.process.X.completed” from “admin”.

```
tdisrvctl –h itditest –op event –e "process.X.completed" –s admin –d "Admin triggered event"
```

Note: All events sent from tdisrvctl using the –e option will be prefixed by “user.”

prop The “prop” option exposes the properties of a config via the TDI-p. It allows the user to get / set / view the properties of a particular config.

The usage for the prop operation is:

tdisrvctl [general_options] **–op prop –c** config_name
[[**-l**] |
[**-o** property_store]
[**-g** key | all] |
[**-s** key=value] [**-e**] |
[**-d** key]]

where:

-c config_name	the name of the config to work with
-l	list all the property stores configured
-o property_store	name of the property store to work with

-g key get the value of the specified key (or keyword 'all' implying get all keys)

-s key=value set the 'key' to the specified 'value'

-e encrypt the value when putting in the store (can be used with –s option only)

-d key delete the specified 'key' from the store.

Notes:

1. The '-l', '-g', '-s', '-d' option are mutually exclusive, and cannot be used together.
2. The '-e' option can only be used with '-s' option.
3. Managing properties stored in the **password store** is NOT supported.
4. While specifying the "-c" option specify the COMPLETE configuration file path on the remote server, or give a path relative to the "configs" folder. To see the relative paths use the "report" option of tdisrvctl:

   ```
   tdisrvctl -op report -l
   ```

Examples:

To see a list of all the property stores for config C1.xml

```
tdisrvctl -op prop -c C1.xml -l
```

To get a list of all the properties for config C1.xml

```
tdisrvctl -op prop -c C1.xml -g all
```

To get a list of all the properties for config C1.xml from store MyStore

```
tdisrvctl -op prop -c C1.xml -o MyStore -g all
```

To set a property MY_PROP to value MY_VALUE for config C1.xml in store MyStore and mark it as protected:

```
tdisrvctl -op prop -c C1.xml -o MyStore -s MY_PROP=MY_VALUE -e
```

queryop

The queryop option returns the list of AL operations exposed in an AssemblyLine.

The usage for the **queryop** operation is:

```
tdisrvctl [general_options] -op queryop -c <configFile> -r <ALname>
```

where

configFile Config file name

ALName Name of the AssemblyLine

Note: While specifying the "-c" option specify the COMPLETE configuration file path on the remote server, or give a path relative to the "configs" folder. To see the relative paths use the "report" option of tdisrvctl:

```
tdisrvctl -op report -l
```

Examples:

To query for *operations* exposed in an AL:

```
tdisrvctl -h itditest -T trust.kdb
  -W secret -op queryop
  -c examples/ADCustomConnector.xml
  -r ADAssemblyLine
```

reload This option can be used to reload running Configs on a particular server.

The usage for reload operation is:

```
tdisrvctl [general_options] -op reload -c [config_list]
```

where:

config_list	comma separated list of Configs to reload

Note: While specifying the “-c” option specify the COMPLETE configuration file path on the remote server, or give a path relative to the “configs” folder. To see the relative paths use the “report” option of tdisrvctl:

```
tdisrvctl –op report –l
```

Example:

To reload Configs C1.xml, C2.xml and C3.xml on remote host itditest:

```
tdisrvctl –h itditest –T trust.kdb –W secret –op reload –c C1.xml,C2.xml,C3.xml
```

report This option can be used for generating a report for a particular config or for listing the configs available on the remote server’s config folder.

The config report will basically list out details of the particular config. The details will be Assembly Lines, Connectors in each assembly line, Connector library, Parser library, Script library, Function Library, Event Handlers, etc. This option will give a one shot view off all the details of a particular config.

The config listing option helps the user in finding out the list of configs available on the remote server and what their exact names are. Of course, only those configs can be seen which are in the “config” folder of the remote server (see global.properties file for property **api.config.folder**). This command cannot obtain list of configs located “anywhere” on the system.

The usage for report operation is:

```
tdisrvctl [general_options] -op report [-c config | -l]
```

where:

-c config	name of the Config whose report is to be generated
-l	the Configs in the remote server’s config folder

Notes:

1. The specified config must be already loaded on the remote server. -
2. Only one of the ’-c’ or ’-l’ option is allowed. Not both. -
3. While specifying the “-c” option specify the COMPLETE configuration file path on the remote server, or give a path relative to the “configs” folder. To see the relative paths use the “report” option of tdisrvctl:

   ```
   tdisrvctl –op report –l
   ```

4. The argument to the -c option is case-sensitive, and must match the name of the config file exactly as known by the server instance, reported by for example "tdisrvctl -op status".

Examples:

To get a complete listing of the details of C1.xml on remote server:

```
tdisrvctl –h remoteserver –op report –c C1.xml
```

To get a list of the configs available in the “config” folder of the remote server:

```
tdisrvctl –h remoteserver –op report –l
```

shutdown

This option can be used to shutdown the TDI server.

The format for this command is:

```
tdisrvctl [general_options] –op shutdown [–o return_code]
```

where:

-o return_code	The return code with which the remote TDI server should exit.

Examples:

To shutdown the local TDI server:

```
tdisrvctl –op shutdown
```

To shutdown the server running on remote host itditest which is configured for SSL (server-auth only)

```
tdisrvctl –h itditest –T trust.kdb –W secret –op shutdown
```

srvinfo

This option is used to display the information of a TDI server.

The format of the command is:

```
tdisrvctl [general_options] –op srvinfo
```

Example:

To view the srvInfo for a TDI server running on localhost

```
tdisrvctl -h localhost –op srvInfo
```

status This option can be used to view status of assembly lines/ event handlers.

The usage for status operation is:

```
tdisrvctl [general_options] -op status -c [config_list | all]
     -r [AL_list     | all]
       -t [EH_list     | all]
```

where:

config_list	comma separated list of Configs or keyword 'all'
AL_list	comma separated list of ALs or keyword 'all'
EH_list	comma separated list of EHs or keyword 'all'

Notes:

1. At least one of the options ('-c','-r' or '-t') must be specified. -
2. The keyword 'all' indicates all configs or AssemblyLines or Event Handlers. -
3. While specifying the "-c" option specify the COMPLETE configuration file path on the remote server, or give a path relative to the "configs" folder. To see the relative paths use the "report" option of tdisrvctl:

   ```
   tdisrvctl –op report –l
   ```

Examples:

To see the status of all configs, ALs and EHs

```
tdisrvctl [general_options] -op status -c all -r all -t all
```

You could also write

```
tdisrvctl [general_options] -op status
```

To see the status of AL1, AL2 and all event handlers:

```
tdisrvctl -h itditest -op status -c c1.xml -r AL1,AL2 -t all
```

Output:

```
 (Component Type # Component Name # RUNNING / STOPPED # Statistics):
1 # AL1 # RUNNING # [get:571] [add:571] [del:3] [requests:2333]....
1 # AL2 # STOPPED #
2 # EH1 # STOPPED #
2 # EH2 # RUNNING #
```

The Component Types are:

- 0 for Config
- 1 for Assembly line
- 2 for Event handler

The Statistics will contain following details (valid for assembly lines only):

- Attribute ″add″ – total number of “add” operations performed
- Attribute ″mod″ – total number of “modify” operations performed
- Attribute ″del″ – total number of “delete” operations performed
- Attribute ″get″ – total number of “getNext” (Iterations) performed
- Attribute ″request” – total number of requests accepted when there is a Server mode Connector in the AssemblyLine.
- Attribute ″callReply″ – total number of “callReply” operations performed
- Attribute ″err″ – total number of errors encountered
- Attribute ″skip″ – total number of ‘skip’ operations performed
- Attribute ″lookup″ – total number of “lookup” operations performed
- Attribute ″ignore″ – total number of “ignore” operations performed
- Attribute ″reconnect″ – total number of “reconnect” operations performed
- Attribute ″exception″ – the exception text if the component terminated with an exception

To see the details of Configs (running and stopped) on a particular server:

```
tdisrvctl -h itditest -op status -c all
```

start This option can be used to start a config / assembly lines / event handlers.

The usage for the start operation is:

```
tdisrvctl [general_options] -op start -c [config]
       -e [password]
          -r [AL_list | all] -alop <alop_Name> [{attr1:value1; attr2:value2;attrn:valuen}] |
       [-f filename
       -t [EH_list | all]
```

where

-c config	name of config to start
-e password	password of config file if it is encrypted
-r AL_list	comma separated list of ALs to start or keyword ′all′
-t EH_list	comma separated list of EHs to start or keyword ′all′

-alop operName — the specific AL operation and list of list required attributes for the specified operation

-f filename — the name of the file where the input attributes and their values are configured for the operation

Notes:

1. The ′-c′ option is mandatory. -
2. The keyword ′all′ indicates all AssemblyLines or Event Handlers. -
3. Required attributes list is mandatory with alop option. -
4. -alop option cannot be used with –r all option. It works only with a specific AL. -
5. While specifying the “-c” option specify the COMPLETE configuration file path on the remote server, or give a path relative to the “configs” folder. To see the relative paths use the “report” option of tdisrvctl:

   ```
   tdisrvctl –op report –l
   ```

Examples:

1. To start assembly line AL1 and AL2, and event handler E1 of config C1 on remote server itditest:

   ```
   tdisrvctl –h itditest –T trust.kdb –W secret –op start –c C1.xml –r AL1,Al2 –t EH1
   ```

   The –r and –t option require that –c option should also be specified. This is because the assembly lines or event handlers mentioned in the command *must* belong to one of the Configs in the –c option.
2. To start assembly line AL1 on remote server itditest with AL operation:

   ```
   tdisrvctl –h itditest –T trust.kdb –W secret –op start
     –c examples/ADCustomConnector.xml
     –r ADAssemblyLine
     –alop myoperation "{ ldapserver:9.182.186.190; suffix:o=ibm,c=us }"
   ```

stop The usage for the stop operation is:

tdisrvctl [general_options] **-op stop -c** [config]
  -r [AL_list | all]
  -t [EH_list | all]

where:

-c config — name of Config

-r AL_list — comma separated list of ALs to stop or keyword ′all′

-t EH_list — comma separated list of EHs to stop or keyword ′all′

Notes:

1. The ′-c′ option is mandatory.
2. While specifying the “-c” option specify the COMPLETE configuration file path on the remote server, or give a path relative to the “configs” folder. To see the relative paths use the “report” option of tdisrvctl:

   ```
   tdisrvctl –op report –l
   ```
3. The keyword ′all′ indicates all AssemblyLines or Event Handlers.
4. The –r and –t option require that –c option should also be specified. This is because the assembly lines or event handlers mentioned in the command *must* belong to one of the Configs in the –c option.
5. The argument to the -c option is case-sensitive, and must match the name of the config file exactly as known by the server instance, reported by for example "tdisrvctl -op status".

Example:

To stop assembly line AL1 and AL2, and event handler E1 of config C1 on remote server itditest:

```
tdisrvctl –h itditest –T trust.kdb –W secret –op stop –c C1.xml –r AL1,Al2 –t EH1
```

tombstone

This option can be used to view tombstone details of previously run configs/ assembly lines/ event handlers.

The usage for tombstone operation is:

```
tdisrvctl [general_options] -op tombstone  -c [config]
      [-r [AL_name] | -t [EH_name]]
         [-age n]
       [[attribute_list] | all ]
```

where:

-age n	display tombstone record for the last ′n′ days (default is 1 day)
-c config	name of Config
-r AL_name	name of AssemblyLine
-t EH_name	name of EventHandler
all	show all tombstone attributes

attribute_list:

-ct	component type
-cn	component name
-guid	tombstone entry′s guid
-et	event type
-ex	exit code
-stime	component′s start time
-ctime	tombstone create time
-desc	error description
-um	user message
-stat	statistics (valid for ALs only)

Notes:

1. The ′-c′ option is mandatory. -
2. While specifying the "-c" option specify the COMPLETE configuration file path on the remote server, or give a path relative to the "configs" folder. To see the relative paths use the "report" option of tdisrvctl:
   ```
   tdisrvctl –op report –l
   ```
3. Only one of the ′-r′ or ′-t′ options is allowed. Not both.
4. The argument to the -c option is case-sensitive, and must match the name of the config file exactly as known by the server instance, reported by for example "tdisrvctl -op status".

Examples:

1. To see the last 2 days tombstone entries (all attributes) for config C1.xml
   ```
   tdisrvctl [general_options] -op tombstone -c C1.xml -age 2 all
   ```
2. To see tombstone entries for config C1 for the past 3 days:
   ```
   tdisrvctl –h itdiserver –op tombstone –c C1 –age 3 all
   ```
3. To see tombstone entries for config C1 for the last 24 hours (specific attributes):
   ```
   tdisrvctl –h itdiserver –op tombstone –c C1 –ct –ctime –cn –um
   ```
4. To see the tombstone entry for AL1 of "rs.xml"

```
tdisrvctl –h itdiserver –op tombstone –c C1 –r AL1
```

Other points to note

- If the user specifies the –T option or the –K option, it means the command line utility must use SSL
- If the user has defined a Solution name for a Config, the -c option should specify the Solution Name.
- If no –h (host) option is specified, the command line interface searches for the environment variable TDI_RSRV. If TDI_RSRV is not set or empty, then it uses "localhost" as default. This is also the case for the –p (port) option: if –p is not specified then it searches for TDI_RPORT, and if that is not specified either it uses the default of "1099".
- The command will return "0" indicating that the command completed successfully without any errors. A "-1" is returned otherwise. For instance a command asking for starting 3 assembly lines will return 0 only if all 3 assembly lines started successfully, otherwise it will return -1.
- The tdisrvctl command line utility will use log4j logging APIs for logging error messages. The log4j configuration file is specified in the startup script (the bat or sh) file. By default the logs are written to <TDI_Install_Dir>/logs/tdisrvctl.log file. The utility must have permission to create and/or write a log file in that location.
- All reported error and warning messages are displayed with an error code prefix. This error code can be used to search the TDI 6.1.1 message guide for an explanation of the error message and operator response.

Chapter 9. Logging and debugging

IBM Tivoli Directory Integrator relies on log4j as a logging engine. It is a very flexible framework that lets you log to file, eventlog, syslog and more, and can be tuned so it suits most needs. It can be a great help when you want to troubleshoot or debug your solution. TDI version 6.0 has additional tracing facilities (discussed in Chapter 10, "Tracing and FFDC," on page 119), though in most cases, the logging functionality described here will suffice.

Some TDI components may have very specific troubleshooting guidelines; always check the particular component's section in the *IBM Tivoli Directory Integrator 6.1.1: Reference Guide* for more information.

The log scheme for the server (ibmdisrv) is described by the file `log4j.properties` and elements of the Config file, while the console window you get when running from the Config Editor (ibmditk) is governed by the parameters set in `executetask.properties` (see "log4j default parameters" on page 117). Logging for the Config Editor program itself is configured in the file `ce-log4j.properties`.

Note: Any of the aforementioned properties files can be located in the Solutions Directory, in which case the properties listed in these files override the values in the file in the installation directory.

You can create your own appenders to be used by the log4j logging engine by defining them in the `log4j.properties` file. You can use drivers built-in to log4j like the default one, which is defined with the statement:

```
log4j.appender.Default=org.apache.log4j.FileAppender
```

The phrase `org.apache.log4j.FileAppender` defines this appender to use the FileAppender class. Additional log4j compliant drivers are available on the Internet, for example drivers that can log using JMS or JDBC. In order to use those, they need to be installed into the IBM Tivoli Directory Integrator installation `jars` directory after which appenders can be defined using those additional drivers in `log4j.properties`. For more information, refer to the log4j project documentation.

In addition to the IBM Tivoli Directory Integrator built-in logging, you can log by adding script code in your AssemblyLine. This is described in much more detail in the *IBM Tivoli Directory Integrator 6.1.1: Users Guide*, in which you will also find out how the interactive debugger works.

Background

Logging and debugging by the system is mainly done through the Task object (the current AssemblyLine). Logging can either be done explicitly (in script) or done by the various components themselves.

Note: The explicit logmsg() calls available to you (i.e., **task.logmsg()** & **main.logmsg()**) can have an optional string parameter indicating the log4j level at which the messages are to be logged. Default is INFO. If the log-level given by the user is invalid for log4j, the message is logged at DEBUG level. Levels include DEBUG, INFO, WARN, ERROR, FATAL.

Logging

Configuring the logging of IBM Tivoli Directory Integrator is done globally (using the files `log4j.properties` which specifies global defaults for Server tasks and `executetask.properties` which specifies defaults for tasks run from the Config Editor) or specifically, using the ibmditk tool, for each AssemblyLine, EventHandler or Config File as a whole. To provide this level of flexibility and customization, the Java Log4J API is used.

Only the parameters that describe how messages are logged are described here.

All log configuration windows operate in the same way: For each one you can set up one or more log schemes. These are active at the same time, in addition to whatever defaults are set in the log4j.properties and executetask.properties files.

Many (but not all) loggers support a Character Encoding option, to control what character set the log files are written in. There are many different character sets; for an informal overview check http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html.

The possible log schemes are as follows:

IDIFileRoller

Sometimes, you want to log to file but keep a limited number of files, as they can fill your disks. IDIFileRoller generates a new file for each run of the Server. The system saves only the specified number of previous logs. If your log is called mylog.txt, and you ask for 2 generations, then after 3 runs you have a mylog.txt (last run) as well as the files mylog.txt.1 and mylog.txt.2, where mylog.txt.2 is the oldest log. From this point, you do not get more files, only newer versions with the same name. Keep two generations of backup files.

IDIFileRoller has the following parameters:

File Path

The name of the file to log to. The path is relative to where you installed IBM Tivoli Directory Integrator The special macro {0} used in filenames is replaced by the name of the Server. Similarly, {1} used in filenames is replaced by a unique identifier generated by the system for you. The {1} macro has no relevance for the special case where you use IDIFileRoller, but is important where you want unique file names.

Number of backup files

If your File Path was mylog.txt, and you select 2 backup-files, the two previous runs have their files renamed to mylog.txt.1 and mylog.txt.2 when you run a third time.

Layout

Determines the format of the log message. Options are:

- Pattern (used if you want to customize the way the messages are logged)
- Simple (format containing just the loglevel and the message)
- HTML (creates an HTML file containing some (relative) time info, thread info, loglevel, category, and message)
- XML (similar to HTML, but generates an XML file (using namespace-prefix log4j))

Pattern

Only used when **Layout** is **Pattern**. See “Creating your own log strategies” on page 118.

Log level

Severity level of the log messages. Options are (from maximum to minimum information):

- DEBUG
- INFO
- WARN

- ERROR
- FATAL

Character Encoding
Character Encoding to be used; like Cp1252, ISO-8859-1, etc.

Log Enabled
Click to enable the use of this Appender.

Console
Logs to the console (standard output). This is in the window where you started the server (ibmdisrv) or the execute task-window in the Config Editor (ibmditk). **Console** has the following parameters:

Layout
See **IDIFileRoller**, previous.

Pattern
See **IDIFileRoller**, previous.

Log level
See **IDIFileRoller**, previous.

Log Enabled
See **IDIFileRoller**, previous.

File Logs to a file. **File** has the following parameters:

File Path
See **IDIFileRoller**, previous.

Append to file
Click to append log information to file. If this is not checked, the file is overwritten.

Layout
See **IDIFileRoller**, previous.

Pattern
See **IDIFileRoller**, previous.

Log level
See **IDIFileRoller**, previous.

Character Encoding
Character Encoding to be used; like Cp1252, ISO-8859-1, etc.

Log Enabled
See **IDIFileRoller**, previous.

Syslog
Enables IBM Tivoli Directory Integrator to log on UNIX Syslog. **Syslog** has the following parameters:

Host name/IP Address
Host to log to.

Syslog Facility
Legal facilities found in the drop-down. Must be supported by the host you are logging to.

Print Facility String
If set, the printed message includes the facility name of the application.

Layout
See **IDIFileRoller**, previous.

Pattern
See **IDIFileRoller**, previous.

Log level
See **IDIFileRoller**, previous.

Log Enabled
See **IDIFileRoller**, previous.

NTEventLog
Enables applications to log using the Windows NT® EventHandler (on Windows platforms). **NTEventLog** has the following parameters:

Layout
See **IDIFileRoller**, previous.

Pattern
See **IDIFileRoller**, previous.

Log level
See **IDIFileRoller**, previous.

Log Enabled
See **IDIFileRoller**, previous.

DailyRollingFile
DailyRollingFile saves old files with a datestamp in their names. It usually is used with the **Append to file** parameter set to **true**. **DailyRollingFile** has the following parameters:

File Path
See **IDIFileRoller**, previous.

Append to file
Create new file or append to existing file, depending on whether this is checked. You usually want this on when using the DailyRollingFile.

Date Pattern
How often the file is rotated. Use the drop-down to choose resolution from minutes to months. For example, if the File Path is set to example.log and the DatePattern set to `'.'yyyy-MM-dd`, on 2003-10-31 at midnight, the logging file example.log is copied to example.log.2003-10-31. Logging for 2003-11-01 continues in example.log until it rolls over the next day.

Layout
See **IDIFileRoller**, previous.

Pattern
See **IDIFileRoller**, previous.

Log level
See **IDIFileRoller**, previous.

Character Encoding
Character Encoding to be used; like Cp1252, ISO-8859-1, etc.

Log Enabled
See **IDIFileRoller**, previous.

SystemLog
This Appender creates log files in a catalog hierarchy under *<TDI_installation_directory>*/ system_logs. For each Config File, there will be a corresponding directory with logfiles named *AL_xxx* or *EH_xxx*, where xxx is the name of the AssemblyLine or EventHandler being run.

This Appender has the following parameters:

Pattern

Specifies the format of the log as defined by LOG4J. The default value is:

```
"%d{ISO8601} %-5p [%c] - %m%n"
```

Additional values available in the field are:

```
"%d{HH:mm:ss} %p [%t] - %m%n"
"%p [%t] %c %d{HH:mm:ss,SSS} - %m%n"
```

Log level

See **IDIFileRoller**, previous.

Character Encoding

Character Encoding to be used; like Cp1252, ISO-8859-1, etc.

Log Enabled

See **IDIFileRoller**, previous.

Log Levels and Log Level control

Log levels can be

- ALL
- DEBUG
- INFO
- WARN
- ERROR
- FATAL
- OFF

ALL logs everything. DEBUG, INFO, WARN, ERROR and FATAL have increasing levels of message filtration. Nothing is logged on OFF.

You can issue log messages to the system or AssemblyLine logs by using the logmsg() method from JavaScript, wherever TDI allows scripting. It can take one or two parameters. See the Javadocs for the logmsg() declaration (package **com.ibm.di.server**, class *AssemblyLine* or class *RS*).

The interface for the logmsg() method (both main and task) with additional log level parameter is **logmsg (String logLevel, String msg)**. The legal values for logLevel are: ″FATAL″, ″ERROR″, ″WARN″, ″INFO″, ″DEBUG″, corresponding to the log levels available for log Appenders. Any unrecognized value is treated as ″DEBUG″.

Note that the IBM Tivoli Directory Integrator logmsg() JavaScript calls log on INFO level by default. This means that setting loglevel to WARN or lower silences your logmsg as well as all Detailed Log settings. However, with the level parameter to the logmsg() call you can override the log level for individual logmsg() calls.

log4j default parameters

These are some of the parameters you find in the files log4j.properties (for ibmdisrv and ibmditk) and executetask.properties (for the Execute Task window that you see in the Config Editor when you run an AssemblyLine from it).

Full documentation can be found at the Apache log4j Project.

log4j.rootCategory= DEBUG, Default

DEBUG is the loglevel for the named Appender (log4j term called Default). If you set this to OFF or level above INFO you do not get output from your script logmessages (see following):

log4j.appender.Default

Defines what type of Appender the named appender Default is. It can be one of the following:

- IDIFileRoller (generates a new file for each run of the Server)
- Console (log to console)
- File (log to file)
- Syslog (log to UNIX Syslog)
- NTEventLog (log to Windows NT EventLog)
- DailyRollingFile (saves old files with a datestamp in their names)
- SystemLog (In a folder structure under *root_directory*/system_logs)

log4j.appender.Default.file

Default log file for File Appender, relative to your installation directory (default ibmdi.log).

log4j.logger.com.ibm.di.*

Log level of various IBM Tivoli Directory Integrator components. Note that, for example, ibmditk shows the log level of the IBM Tivoli Directory Integrator Config Editor itself (not the processes you are running inside it). Do not change these.

Creating your own log strategies

You can use this framework to differentiate how the different AssemblyLines and EventHandler log.

Note: This information is intended for users who want to continue using global.properties file to customize logging output. You can customize logging output through the Config Editor (ibmditk).

The following section defines a log scheme called CONSOLE, and that can later be used by specific AssemblyLines or EventHandlers:

```
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=%d [%t] %-5p - %m%n0
```

Now in order to have the AssemblyLines and EventHandler myAL and myEH, you need the lines:

```
log4j.logger.AssemblyLine.myAL=INFO, CONSOLE
log4j.logger.EventHandler.myEH=INFO, CONSOLE
```

Refer to the full log4j (version 1.2) documentation for description of the ConversionPattern parameters. Here are some parameters:

%d Date/time depending on format.

%p Priority.

%c Category.

Note: this is typically in the form *Type.alName.xxx*. *Type* can be EventHandler or AssemblyLine, *alName* is the name of the AssemblyLine (or EventHandler as name by the creator), and *xxx* is a unique ID for the thread. **%c{2}** outputs *alName*.

%m Message.

%n Newline.

%t Threadname.

Chapter 10. Tracing and FFDC

In addition to the user-configurable logging functionality described in Chapter 9, “Logging and debugging,” on page 113, IBM Tivoli Directory Integrator is instrumented throughout its code with tracing statements, using the JLOG framework. This is a logging library similar to log4j, but which is used inside TDI specifically for tracing and First Failure Data Capture (FFDC). To which extent this becomes visible to you, the end user, depends on a number of configuration options in the global configuration file `jlog.properties`, and the Server command line option **-T**.

Note: Normally, you should be able to troubleshoot, debug and support your solution using the logging options described in the chapter, Chapter 9, “Logging and debugging,” on page 113. However, when you contact IBM Support for whatever reason, they may ask you to change some parameters related to the tracing functionality described here to aid the support process.

Understanding Tracing

Tracing is done in TDI’s code using JLOG’s PDLogger object. PDLogger or the **P**roblem **D**etermination Logger logs messages in Logxml format (a Tivoli standard), which IBM Support understands and for which they have processing tools.

The basic level of information traced, as handled by the PDLogger APIs, is:

```
Date | Time | ClassName | methodName | MachineName | IP | {Entry/Exit/Exception} | [Parameter]
```

(The "`|`" character serves a documentation purpose only, it is not part of the actual log.)

Tracing is not performed using log4j Appenders for the following reasons:

1. Trace is always to be enabled
2. You wouldn’t want multiple traces enabled in the server (could be several for each AL if Appenders were used).

The PDLogger is attached to the JLOG **SnapMemory** handler and the **JlogSnapHandler**.

The **SnapMemory** Handler logs trace messages to memory. On the trigger of a LogEvent (eg: occurrence of a specific Log level Trace message, as defined by the `jlog.levelflt.level` filter, or an application crash or on the occurrence of a specific TMS XML messageID) the Trace memory buffer is written to a file by the **JlogSnapHandler**.

To make Tracing and Log messages in TDI unique across all IBM products, they are prefixed with a unique Message Prefix: **CTGDI**.

All error messages are prefixed with a unique TMSXML messageID which indicates the cause of the error and an operator response.

All info messages are also prefixed with a unique TMSXML messageID which may or may not provide the operator response.

Configuring Tracing

The jlog.logger.level property in the `jlog.properties` file can be used to set the desired trace level. The trace level can be set to any of the following JLOG log levels: (Hierarchy, from most severe to least severe)

- FATAL

- ERROR
- WARNING
- INFO
- DEBUG_MIN
- DEBUG_MID
- DEBUG_MAX

The default level is DEBUG_MIN.

The trace level and other properties can be changed by dynamically setting the right Java property from scripts within TDI. The script LogCmd.bat (Windows) and LogCmd.sh (Unix), present in the installation directory, can be used to set trace properties dynamically.

Note: The JLOG logger starts a command server on port 9992 (default) to listen to log commands sent by the LogCmd command line utility. For the logcmd scripts to work, the command server needs to be started first; this is controlled by the jlo.noLogCmd property in `jlog.properties`. By default (for security reasons), in TDI 6.1.1 the command server is disabled, that is, the jlog.noLogCmd property is set to **true**. To use the logcmd scripts you need to first set the jlog.noLogCmd=false in the `jlog.properties` file, and restart the Server. However, even with the command server running and listening, it will only answer requests issued from the local machine.

Refer to the comments in the `jlog.properties` file for further guidance on JLOG configuration options.

Useful JLOG parameters

Property	Value	Description
jlog.snapmemory.queueCapacity	Default 10000	The number of logevents that can be stored in the snapmemory handlers queue.
jlog.snapmemory.dumpEvents	true	If set to true, the handler will immediately send all the queued events to its output listeners. The property will then be reset to false.
jlog.snapmemory.userSnapDir	CTGDI/FFDC/user/	The directory to place the trace dump file when a user triggers an FFDC action by using the logcmd scripts.
jlog.snapmemory.isSync	Default false	If ″true″ log events will be dumped to the snap shot file synchronously. This does not spawn a new thread, and causes the logger to block until the snapshot is complete.
jlog.snapmemory.userSnapFile	userTrace.log	
jlog.snapmemory.triggerFilter	jlog.levelflt	The level filter to be used to take JFFDC action.
jlog.snapmemory.msgIds	*E	The TMSXML message filter to be used for JFFDC action.
jlog.snapmemory.mode	PASSTHRU or BLOCK. Default is PASSTHRU.	Indicates whether to pass the listed Ids under the msgIDs property to the filter or block them.
Jlog.snapmemory.msgIDRepeatTime	10000 (in milliseconds)	The minimum time in milliseconds, after passing a logEvent with a given TMS message ID, before another logEvent with the same id can be passed.

The default value for jlog.snapmemory.triggerFilter sets up a trigger filter named jlog.levelflt. An attribute of such a filter is the message severity, which takes one of the JLOG Log values as described above. By default, the entries

```
jlog.levelflt.className=com.ibm.log.LevelFilter
jlog.levelflt.level=FATAL
```

will set up the FFDC code to cause the memory buffer to be dumped to the trace log when a trace message of severity FATAL occurs. The jlog.levelflt.level property can take any of the other Log level values as well, but only values of ERROR or FATAL will make much sense as otherwise the amount of FFDC dumping will be very high, causing huge slowdowns of the TDI Server.

Chapter 11. Administration and Monitoring

IBM Tivoli Directory Integrator (TDI) 6.1.1 bundles a fully supported, Web-based Administration and Monitoring Application (AMC). The AMC can be used to remotely start, stop and manage TDI Configs and AssemblyLines.

IBM Tivoli Directory Integrator 6.1.1 also ships an Action Manager (AM) with the AMC. The Action Manager is a stand-alone Java application that interacts with the AMC database and uses the TDI Remote Server API to manage remote AssemblyLines.

The Administration and Monitoring Console is comprised of a Java WAR file (tdiamc.war) that can be deployed on any J2EE compliant Web Server.

The current version bundled with IBM Tivoli Directory Integrator 6.1.1, AMC version 3, supports TDI 6.1.1, TDI 6.1 and TDI 6.0 – the latter two products with some restrictions (see below). Any older versions of TDI are not supported.

Note: IBM Tivoli Directory Integrator 6.1.1 and solutions developed and deployed with it can also be monitored with IBM Tivoli Monitoring (ITM) Server and Portal, by virtue of TDI's Java Management Extension (JMX) interface. You will find an example as to how this can be accomplished in the <TDI_install_folder>/examples/Tivoli_Monitoring directory.

Installation and Configuration

Installing AMC on Embedded WAS Express

Note: These instructions require that you be familiar with theIBM Tivoli Directory Integrator 6.1.1 Installation procedures. See “Using the platform-specific TDI installer” on page 10 for information about TDI Installation. Installing AMC also installs AM.

IBM Tivoli Directory Integrator 6.1.1 ships a lightweight version of WebSphere Application Server (WAS) Express (except on i5/OS where a system-provided WAS will be used).

The TDI Installer automatically installs Embedded WebSphere Application Server Express and deploys AMC on it if you select those options during IBM Tivoli Directory Integrator 6.1.1 installation.

To install and deploy the Administration and Monitoring Console on the Embedded WebSphere Application Server Express:

1. Invoke the TDI 6.1.1 installer.
2. During installation, select the **Custom** install. (Typical installation does not offer AMC option.)
3. On the "Select Features" panel of the installation, select **AMC: Administration and Monitoring Console** and **Embedded version of WebSphere Express v6.0.2** .
4. Finish installing TDI 6.1.1.

Selecting the AMC with embedded Websphere option installs WAS, creates a profile called **amcprofile** and deploys the tdiamc.war file into this profile.

Note: There is a problem with embedded WAS on installation and post installation due to issues with the JRE shipped with embedded WAS. If installing AMC and the embedded WAS RHEL 3 on zLinux, then the JIT needs to be disabled before running the installer from the command line:

```
export JAVA_COMPILER=NONE
```

In the same command line window, execute the installer and perform a normal installation. Next, download and apply the JRE SR4-1 update available at http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg24012074. This should be applied to the JRE in the embedded WAS located at *<TDI__Install_Dir>/AppServer/java/jre* . After this point, the JIT no longer needs to be disabled.

The default location of the AMC is:

```
<TDI_Install_Dir>/AppServer/profiles/amcprofile/installedApps/DefaultNode/tdiamc.war.ear/tdiamc.war/
```

Deploying AMC as a Windows Service by means of WASService.exe

WAS provides a command called WASService.exe in the ″bin″ directory which allows you to create a windows service for any profile.

With regards to AMC, the command to run AMC as a windows service (automatically) would be (run this command from the <TDI_install>/AppServer/bin directory):

```
WASService.exe -add "MyServiceName" -servername server1 -profilePath "..\profiles\amcprofile" -startType automatic
```

For full details on this command, see http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/rins_wasservice.html

Installing AMC on an existing WAS 6.0 or WAS 6.1 version

To install the Administration and Monitoring Console on an existing WAS setup:

1. Obtain the tdiamc.war file.
2. Choose the WAS profile into which AMC must be deployed, or create a new one.
3. Deploy the tdiamc.war file to the chosen profile.
4. Start the profile.

The IBM Tivoli Directory Integrator 6.1.1 installer can be used to deploy AMC on an existing WAS instance automatically, or you can choose to deploy AMC manually using the scripts shipped with TDI 6.1.1 or the command line utilities shipped with WAS or using the WAS Application deployment tool.

Installing AMC on WAS using the TDI 6.1.1 Installer

1. Invoke the IBM Tivoli Directory Integrator 6.1.1 installer.
2. Select the **Custom** install. Typical installation does not offer AMC option.
3. On the "Select Features" panel:
   - Select **AMC: Administration and Monitoring Console**.
   - Do not select **Embedded version of WebSphere Express v6.0.2**.
4. Because the Embedded version of WebSphere Express option was not selected, the WebServer selection panel is shown. This panel automatically detects existing WAS instances. Select one of the following options:
   - You can select a currently installed WebSphere Application Servers, if any are detected, by selecting the **Detected WebSphere Application Server** radio button and selecting the desired server from the drop-down menu.
   - You can create a custom location for the WebSphere Application Server by selecting the **Custom location of WebSphere Application Server**. Use the **Browse** button to navigate to the desired location.
   - To manually deploy AMC at a later time, select **Do not specify. I will manually deploy AMC at a later time.**
5. After having selected the WAS instance to install, the "Profile Selection" panel is shown, displaying various existing WAS profiles. Select the profile into which you want the AMC installed.
6. Finish the installation. Administration and Monitoring Console is installed on the selected WAS instance and profile.

Installing AMC on WAS using the TDI 6.1.1 Scripts or WAS Commands

1. Invoke the IBM Tivoli Directory Integrator 6.1.1 installer.
2. Select the **Custom** install. Typical installation does not offer an AMC option.
3. On the "Select Features" panel:
   - Select **AMC: Administration and Monitoring Console**.
   - Do not select **Embedded version of WebSphere Express v6.0.2**.
4. Because you did not select the embedded version of WAS Express, the "WebServer Selection" panel will be shown. This panel automatically detects existing WAS instances. Select **I will manually deploy AMC at a later time**.
5. Finish the installation. The installer places the tdiamc.war file in the `<`*`TDI_Install_Dir`*`>/amc` directory and the AMC scripts in the `<`*`TDI_Install_Dir`*`>/bin/amc` directory.
6. If you wish to create a new profile in WAS, execute the `createProfile` command in the <*TDI_Install_Dir*>/bin/amc/ folder.

   You must modify this script to specify the WAS install location and the WAS profile name to create: Set the `APPSRV_INSTALLROOT` variable to the WAS install location, and `AMC_PROFILE` to the profile name. This is a wrapper script that calls the `wasprofile` command of WAS to create a profile.
7. To deploy the tdiamc.war file in WAS, you must edit the install script in the <*TDI_Install_Dir*>/bin/amc folder. Specify your WAS install location and the WAS profile name to deploy AMC into. Set the `APPSRV_INSTALLROOT` variable to the WAS install location, and `AMC_PROFILE` variable to the profile. This is a wrapper script which calls the wsadmin command of WAS.

Starting and stopping AMC following installation

To start the Administration and Monitoring Console:

- If you installed AMC using the embedded WAS Express, invoke the following command from the <*TDI_Install_Dir*>/AppServer/bin directory:

  ```
  startServer server1 –profileName amcprofile
  ```
- If you installed AMC using an existing WAS 6.0 or 6.1, invoke the following command from the <*WAS_Install_Dir*>/bin directory:

  ```
  startServer server1 –profileName <selectedProfile>
  ```

Note:

> You can also start and stop the AMC by running the following scripts shipped in <*TDI_Install_Dir*>/bin/amc folder:
>
> - To start AMC, run the `start_tdiamc` script.
> - To stop AMC, run the `stop_tdiamc` script.

Once the Administration and Monitoring Console is started, you can access it from the following URL: http://localhost:<*port*>/tdiamc

If you installed AMC using an existing WAS 6.0 or 6.1, stop the AMC by running the following command from the <*WAS_Install_Dir*>/ bin directory:

```
stopServer server1
```

If you installed AMC using Embedded WAS Express, stop the AMC by running the following command from the <*TDI_Install_Dir*>/AppServer/bin directory:

```
stopServer server1
```

Installing AMC on Tomcat 5.0.x

To install the Administration and Monitoring Console on Tomcat 5.0.x, download and install Tomcat 5.0.x from http://jakarta.apache.org.

Ensure that tomcat has been successfully installed by pointing your browser to http://localhost:<*port*>/

To install the AMC on Tomcat all that is required is the tdiamc.war file that is shipped with TDI 6.1.1 in the <*TDI_Install_Dir*>/amc folder.

Copy the tdiamc.war file and drop it into the <*Tomcat_Install_Dir*>/webapps folder and restart Tomcat.

AMC will automatically be deployed and installed when Tomcat is started.

The URL that launches the Administration and Monitoring Console is: http://localhost:<*port*>/tdiamc

Notes:

1. The tdiamc.war file will be placed into the folder by the installer only if you chose a Custom install option during installation, and selected the **AMC: Administration and Monitoring Console** option on the "Select Features" panel.
2. In <Tomcat_Install_Dir>/common/endorsed/ the **xercesImpl.jar** must be replaced by IBM JVM's **xml.jar**. This file can be found in your TDI installation at the following location: <TDI_Install>/jvm/jre/lib/xml.jar. This is necessary since the TDI Server API uses the IBM XML parser which must be present at the client end for being able to correctly serialize data. Do not delete the **xercesImpl.jar** but back it up in a safe location.

Note:

Backward Compatibility with previous versions of TDI

When AMC for TDI 6.1.1 is used to manage older versions of TDI, some functionality will be limited.

- The concept of TDI Properties was changed in TDI 6.1. For this reason, AMC will not support editing/viewing/management of properties when working with a TDI 6.0 server. Also, the properties related triggers and actions will be unavailable when working with a remote TDI 6.0 server.
- The **sendEventNotification** API for sending events remotely to a TDI Server was introduced in TDI 6.1. Therefore AMC will not support sending of TDI events to remote TDI 6.0 servers. The Send Event Notification Action (in AM Configuration) panels will not be available when working with a TDI 6.0 server.
- The Tombstone Manager Feature was introduced in TDI 6.1. Hence viewing of tombstones, and seeing the last run time/stop time of an AL is not possible for remote TDI 6.0 servers. In addition, the "View Tombstones in chronological order" feature (and view last stop time/run time) was added in TDI 6.1.1, by adding an API in Tombstone Manager. Hence, this feature is available only in TDI 6.1.1.
- The remote config folder and the ability to view Configs in the remote config folder was a feature introduced in TDI 6.1. For this reason, in the "load-reload Configs" panel, only those Configs will be shown which are already loaded on the remote TDI 6.0 server. A Load operation will not be available for TDI 6.0 servers. Only Stop and Re-load operations are available.
- The AL Operation feature was introduced in TDI 6.1. Therefore starting an AL with AL Operation will not be supported for TDI 6.0 servers.
- You are able to create a Config view (minus properties), select ALs to expose, select health AL, configure users, start/stop ALs, stop/reload Configs, create rules on ALs when working with a TDI 6.0 server.
- Custom Authentication (using ldap or JavaScript) was introduced in TDI 6.1. AMC's add server panel supports username and password fields for this purpose. If a user attempts to pass username and

password to a TDI 6.0 server, AMC will not be able to connect to the remote TDI server. There is no way for AMC to find out that the remote server is TDI 6.0, since it cannot connect with “incorrect” settings.

- Creation of Quick config view:
  - Publish Solution: this feature is not available with TDI 6.0 and TDI 6.1 servers .
  - Create Config View with All Assembly lines exposed: this feature is available with TDI 6.0, 6.1 and 6.1.1 servers.
  - Create Config View with All Assembly lines and All properties exposed: this feature is not available with a TDI 6.0 server.
  - In the View Tombstone panel only the 30 most recent tombstones entries will be displayed.

Enabling AMC

The configuration file for the Administration and Monitoring Console is the `amc.properties` file that is located in the `tdiamc/` directory at the same level as the `WEB-INF` directory. This file contains the AMC’s database configuration properties, LDAP properties, SSL related properties and help server details.

The Administration and Monitoring Console makes use of Cloudscape version 10 ("Derby") to store data. When AMC is started for the first time, AMC creates a `tdiamcdb` folder inside the Web Server directory and creates the tables needed for AMC to function. The Cloudscape database can be accessed in either the network mode or embedded mode. By default, AMC is shipped with Cloudscape configured in network mode. The following properties in `amc.properties` are an indication of Cloudscape configured for network mode:

```
com.ibm.di.amc.jdbc.database=jdbc:derby://localhost:1527/tdiamcdb;create=true
com.ibm.di.amc.jdbc.driver=org.apache.derby.jdbc.ClientDriver
com.ibm.di.amc.jdbc.urlprefix=jdbc:derby:
com.ibm.di.amc.jdbc.user=APP
com.ibm.di.amc.jdbc.password=APP
com.ibm.di.amc.jdbc.start.mode=automatic
com.ibm.di.amc.jdbc.host=localhost
com.ibm.di.amc.jdbc.port=1527
com.ibm.di.amc.jdbc.sysibm=true
```

The property `com.ibm.di.amc.jdbc.database` points to Cloudscape in network mode, running on localhost:1527. The database name being accessed is `tdiamcdb`, and `create=true`, indicating that AMC will create the database if not found. It is recommended that the `create=true` be set to false later, so that in case the database path gets modified, AMC does not re-create the database, but instead throws a database not found exception. It is also suggested, that database be set to an absolute path to avoid any confusion about the database path later.

The Administration and Monitoring Console will automatically attempt to start the Cloudscape server in network mode, by reading the `com.ibm.di.amc.jdbc.host` and `com.ibm.di.amc.jdbc.port property`. Specify `localhost` if you do not want any remote connections to be allowed to this database, otherwise specify the IP address of the local machine.

Note: Sometimes AMC cannot start the Cloudscape database in network mode because of security restrictions by the WAS container. For example, in WAS 6.0.1, there is a `startNetworkServer` script in the `<TDI_Install_dir>/bin/amc` folder that must be run before starting AMC. Meanwhile, the `tdiamcdb` database must be in directory `<TDI_Install_dir>/AppServer/derby`.

The Administration and Monitoring Console can also be configured to connect to the Cloudscape database in Embedded Mode. In this case, the Action Manager (a separate application that also talks to the AMC database) will be unable to connect to AMC’s database. This is because in Embedded Mode, only one JVM at a time is allowed to connect to the Cloudscape database. If Action Manager is not needed, then it is recommended to run AMC’s Cloudscape database in Embedded Mode. The following example shows the `amc.properties` file with Cloudscape configured for embedded mode:

```
##Location of the database (embedded mode)
com.ibm.di.amc.jdbc.database=tdiamcdb
com.ibm.di.amc.jdbc.driver=org.apache.derby.jdbc.EmbeddedDriver
com.ibm.di.amc.jdbc.urlprefix=jdbc:derby:
com.ibm.di.amc.jdbc.user=APP
com.ibm.di.amc.jdbc.password=APP
```

The `com.ibm.di.amc.jdbc.database property` points to the location of the AMC database. We suggest that this value be set to an absolute path to avoid any confusion about the database path later.

AMC Logs

The Administration and Monitoring Console logs are stored in the `auiml/logs` directory inside the Web container. The AMC.log.0 file contains the most recent logs.

The configuration of AMC logs can be done by modifying the `WEB-INF/classes/logging.properties` file. AMC logging follows the Java logging standard (java.util.logging). The logs can be viewed from within Administration and Monitoring Console by clicking the **Logs** link on the top right hand corner of any AMC panel.

Action Manager

The Action Manager (AM) is a standalone Java application that allows you to monitor multiple TDI Servers and AssemblyLine execution using user-defined rules, triggering conditions and actions defined in AMC. The Administration and Monitoring Console (AMC) has an AM Configuration panel that allows users to configure various Action Manager rules.

A rule is a combination of a Trigger type and a set of associated actions. A rule specifies that when a Triggering condition is detected, then the associated set of actions must be executed. The various trigger types available in AMC are described below:

Table 6.

Trigger Type	**Trigger Details**	**User Input for trigger**
No trigger	A rule with this triggering type has no triggering condition, and hence will never get triggered by itself. The only way this rule can be executed is if some other rule executes this rule	No details required
On AL termination	A rule with this triggering type will get triggered when the Action Manager receives an AL termination event for this particular AL.	AssemblyLine name
Time since last execution	A rule with this triggering type will get triggered when the Action Manager detects that the specified assembly line has not run for the specified period. Note: This rule will be triggered only once. After that Action Manager will wait for receiving a Start AL event before resetting the Rule back to Ready mode. This is done so that the rule does not repeatedly get triggered for a single occurrence of the triggering condition.	″AssemblyLine name″, ″Not Run Since″ and ″Unit″.

Table 6. (continued)

Trigger Type	Trigger Details	User Input for trigger
On query AL result	A rule with this triggering type is triggered when the last ″work″ entry of the specified AL, contains the specified ″Attribute″ matching the given ″condition″ and ″value″. This condition will be checked only when the Action Manager receives a Stop AL event. **Note:** When a trigger of this type is used, the AL should not be a short running AL. This is because AM stores the handle of the Al object on receiving the Start AL event. Later on receiving the Stop AL event, AM uses this handle to query the final work entry attributes. If the AL terminates before AM can store the handle, then AM will not be able to query the work attributes. Usually an execution time of 10 seconds will be sufficient (this can be achieved by putting a system.sleep(10) before the AL terminates, for example in the **epilog** hook).	″AssemblyLine name″, ″Attribute″,″Condition″,″Value″.
On server API failure	A rule with this triggering type will be triggered when the Action Manager is unable to connect to the remote server using the Server API.	No details required
On received Event	A rule with this triggering type will be triggered when the Action Manager receives an event which satisfies the criteria mentioned. **Note:** If any of the criteria is to be ignored, just leave it blank.	″Event type″, ″Event Source″, ″Event Data″. Event Data is optional. Event type or source – one of them must be specified.
On Property	A rule with this triggering type will get triggered when the specified property meets the specified condition. The Action Manager periodically checks for this property. **Note:** This rule gets triggered only once, and gets reset back to ready state only when Action Manager detects that this property does not meet the specified criteria any longer. This is done so that the rule does not repeatedly get triggered for a single occurrence of the triggering condition.	″Property Name″, ″Condition″, ″Value″.

When a rule gets triggered, the Actions associated with the rule are executed by the Action Manager sequentially. The following are the various types of Actions that are available in AMC:

Table 7.

Action	Action Details	User Input for action
Start AssemblyLine	This action starts the specified AL of the specified config file on the specified TDI server. The Config field should mention the complete path of the configuration on the remote server. The Config Password field is optional and is required only if the remote config is password protected.	″AssemblyLine″, ″Of Configuration″, ″On Server″, ″Config Password″.
Stop AssemblyLine	This action stops the specified AL of the specified configuration on the specified TDI Server. The Config field should mention the complete path of the configuration on the remote server.	″AssemblyLine″, ″Of Configuration″, ″On Server″.
Enable/Disable AM Rule	This action will Enable or Disable the chosen AM rule.	″RuleName″ ″State″

Table 7. (continued)

Action	Action Details	User Input for action
Execute AM Rule	This action will cause the execution of the specified rule, which will in-turn imply execution of all the actions specified in that particular rule.	″RuleName″
Notify Event	This action will cause the Action Manager to emit an event with the specified details to the Server associated with the current config view. See the Session.sendCustomNotification() API for details.	″Event type″, ″Source″, ″Data″.
Modify Property	This action will cause the Action Manager to modify the selected property based on the specified operation.	″Property″, ″Operation″, ″Value″.
Copy Property Value	This action will cause the Action Manager to copy the value of the Source property to the Destination property.	″From Property″, ″To Property″.
Write to Log	This action will cause a log of the specified Severity/Message/Description to be logged into the Action Manager logs and the AMC database. The same log is shown when the user goes to the Monitor Status -> Config View Details -> AM Results table. It is advised to always have at least one Log action (containing descriptive text) in every rule.	″Severity″, ″Message″, ″Description″.

Rules that are configured for Config views in AMC, are stored in AMC's Cloudscape Database. When the Action Manager is run, it connects to the AMC database in network mode, reads the Action Manager-related tables, and creates threads in memory for every AM rule specified. Each of these threads listens/polls for its respective triggering conditions. The moment any thread detects the occurrence of its respective triggering condition, it queries the database for the set of actions associated with the rule, and executes them sequentially.

The Action Manager runs the following threads in addition to the rule threads that are listening for trigger conditions:

1. HealthAssemblyLine – The Health AssemblyLine thread periodically triggers the Health ALs for querying the status of the solutions, and logging the status back into the AMC database. The health AL must store the status in the ″healthAL.result″ and ″healthAL.status″ attributes of their final work entry.
2. ServerStatusListener - The ServerStatusListener thread is created for every server registered with AMC. This server checks for the server accessibility. If the server has become inaccessible, all rules threads created for the server will be terminated (except for those with triggering type 'On Server API failure'). Similarly if the server becomes accessible, rule threads will be created for any rules associated with this server.
3. ConfigLoadReloadListener – The ConfigLoadReloadListener thread is created for every running server registered with AMC. It is registered to the remote server for any config load unload events. Rule threads will be appropriately terminated, created or refreshed depending on the config event.
4. ServerModificationListener – The ServerModificationListener thread checks for any updates to the set of servers registered in AMC. Depending on the type of change (added, removed, etc.) rule threads will be terminated, created or refreshed.
5. ActionManagerStatusUpdate – The ActionManagerStatusUpdate thread is used to update AMC on whether the Action Manager is currently running or not.
6. DatabaseModificationListener - This database listener thread continuously monitors addition, modification or deletion of rules. Whenever any changes in the rules are detected, the AM threads are added/recreated appropriately at runtime.

The Action Manager also updates the AMC database with its run details. Whenever an Action Manager rule is triggered, Action Manager logs an entry into the AMC database, registering the rule name that got triggered, and the triggering time. Also, if any AM Log action is configured for the AM rule, then that also gets logged into the AMC database. These database entries are used to show the appropriate status in Monitor Panels of AMC.

Enabling AM

The Action Manager is installed in the `<TDI_Install_Dir>/bin/amc/Action Manager` folder. It contains the following files:

- am_logging.properties - This file controls Action Manager logging properties. Just like AMC, it also follows the java.util.logging logging standard.
- am_config.properties - This is the configuration file for the Action Manager.
- testadmin.jks - This is the ActionManager's truststore and keystore file.

  Note: This is a sample truststore and keystore file; for added security, you should generate your own.

The Action Manager connects to AMC's Cloudscape database using the Network Mode driver.

The following properties (in am_config.properties) must point to the Administration and Monitoring Console's database:

```
com.ibm.di.amc.am.jdbc.database=jdbc:derby://localhost:1527/tdiamcdb;create=false
com.ibm.di.amc.am.jdbc.driver=org.apache.derby.jdbc.ClientDriver
com.ibm.di.amc.am.jdbc.urlprefix=jdbc:derby:
com.ibm.di.amc.am.jdbc.user=APP
com.ibm.di.amc.am.jdbc.password=APP
com.ibm.di.amc.am.jdbc.start.mode=automatic
com.ibm.di.amc.am.jdbc.sysibm=true
com.ibm.di.amc.am.jdbc.networkserver.host=localhost
com.ibm.di.amc.am.jdbc.networkserver.port=1527
```

To start the Action Manager, run the `startAM` script stored in the `<TDI_Install_Dir>/bin/amc` folder. When the Action Manager is started, it attempts to connect to AMC's database. If it fails in performing any initial setup tasks, it will exit with an exception message. Check the `am_config.properties` file to ensure it points to the correct database. If the database settings appear to be correct, then ensure that the database that Action Manager is attempting to connect to is running in network mode and that AMC can connect to the same database. You may use the `startNetworkServer.bat(sh)` to start the Cloudscape DB in network mode.

Further configuration of run-time rules, triggers and actions is described under "AM Configuration" on page 147.

AMC and AM Security

Introduction

AMC is a web based application for monitoring and managing remote TDI solutions. AMC makes use of the TDI Remote Server API to communicate with TDI. For this reason, all the security restrictions and configuration settings that are applicable to TDI Remote Server API clients (as mentioned in previous sections) are valid for AMC too. Besides those, AMC also provides certain in-built security features.

Action Manager is installed along with AMC. Action Manager configures itself and behaves based on rules set in the AMC DB by AMC Users. To monitor remote ALs and to take action based on configured rules, Action Manager, just like AMC, makes use of the TDI Remote Server API to communicate with TDI servers.

AMC and SSL

Multiple TDI Servers can be registered with AMC. Each TDI server may be configured differently; one TDI server could be running with SSL off, one with SSL on, one with Custom Authentication on and SSL on – and various other combinations. AMC can be used to connect and administer any of these servers simultaneously. As mentioned earlier, to configure TDI to run in SSL mode the **api.remote.ssl.on** property should be set to **true** in `global.properties` (or `solution.properties`).

AMC being a web application running inside a Web Container – automatically will inherit some properties and security restrictions from the Web Container. For instance, if the Web Container has an SSL key store or SSL trust store configured, then that would be automatically applicable to AMC. But AMC can also override that – and specify its own key store and trust store.

For being able to communicate with TDI Remote Server API running on SSL, AMC must have a key store configured which contains the certificate that is trusted by the TDI remote Server API (i.e it must be present in TDI's trust store's trusted certificates section) and AMC must have a trust store configured which contains the certificate that is sent by the TDI remote Server API. In other words – the certificate that is present in TDI server's key store must be present in AMC's trust store and the certificate that is present in TDI trust store must be present in AMC's key store.

For example, the default installation of TDI is shipped with certain stores (jks files). When you run TDI in SSL mode, then to connect to AMC its key store and trust store must both be set to the same value: <TDI_installation_directory>/serverapi/testadmin.jks and the password being "administrator". Since testadmin.jks contains both trusted certificates and signer certificates – a connection gets established. It is recommended to use your own SSL key stores and trust stores.

In AMC, the path of the Trust Store and Key store can be set by logging into AMC as "superadmin" (Console Administrator) and navigating to the following panel: **Console Administration->Manage Console Properties->SSL settings**. The settings for trust store and key store are written to **amc.properties** file inside the tdiamc folder in Web Container. You can alternatively choose to edit the **amc.properties** file directly. The Trust store and Key store values are set to `testadmin.jks` by default. This testadmin.jks file is present in the `<TDI_Install_Dir>\AppServer\profiles\amcprofile\installedApps\DefaultNode\tdiamc.war.ear\tdiamc.war\` directory. The keystore and truststore password are set to "administrator" by default. To establish an SSL based connection with a remote TDI server, you need to start the server in "SSL enabled" mode, and for a Non SSL based connection, start the server in "SSL disabled" mode. There is no need to configure SSL settings in AMC normally; default SSL settings are provided.

For each TDI server running over SSL that you wish to register with AMC, you must import the necessary certificate into AMC's trust store and the necessary AMC's key certificate into TDI's trust store. The idea here is that AMC must trust TDI and TDI must trust AMC to be able to make a secured two-way SSL connection.

Since AMC runs inside a Web Container, the URL for AMC will be *http://hostname:port/tdiamc*

This indicates that AMC is running on HTTP. If you wish to run AMC on Secured HTTP (HTTPS), then your Web Container (WebSphere, Tomcat, etc) must be configured to open an SSL port for web communication. See the Web Server's documentation on how to configure it for SSL. In that case, AMC's url will change to: https://hostname:secured_port/tdiamc

Action Manager (AM) monitors running Configs and AssemblyLines on remote TDI Servers based on rules configured in AMC. Action Manager ships with the key store and trust store required to connect to a remote TDI server. The SSL properties are defined in the `am_config.properties`. See details on how to configure AMC for SSL in previous sections - the same is applicable for Action Manager.

AMC and Remote TDI Server

AMC can connect to multiple TDI Server's remotely. Each TDI Server can be configured in one of the following ways:

- Non SSL
- SSL
- Custom Authentication with Non-SSL
- Custom Authentication with SSL

Let's discuss each of these cases in detail.

When a remote TDI server is configured for Non SSL (i.e api.remote.ssl.on=false) then the key store or trust stores of AMC do not come into play, even if correctly configured – since no SSL connection is being attempted. In this case the AMC Server's machine IP address must be registered with the TDI server. This is done by editing the `global.properties` (or `solution.properties`) file. The property to update is: **api.remote.nonssl.hosts**. Once the AMC machine's IP address is entered in the global.properties file of the remote TDI server, AMC will be able to connect to that particular server. It's a way of saying – I trust remote server connections (AMC connections) from only those machines whose IP addresses I have mentioned in my **api.remote.nonssl.hosts** property.

Note: If the TDI server is running on the same machine as AMC, then editing this property is not required.

When a remote TDI server is configured for SSL (i.e **api.remote.ssl.on=true)**, then the SSL key store and trust store for AMC must be setup appropriately.

For details on this, see the previous section on AMC and SSL. In addition to being configured for SSL or Non-SSL, a remote TDI server may also require Custom Authentication – in which a username and password need to be passed while making a connection to the remote TDI server. The remote TDI server will validate this user name and password against some 3rd party repository like LDAP, file, database, script, etc and then make a decision on whether to allow the Server API client to make a connection or not. In such cases, while registering a server with AMC (**Manage TDI Servers->Add / Edit TDI Server**) in the Authentication Mode panel – select "**Custom or LDAP Authentication**" and mention the Username and Password which AMC must pass every time it attempts to connect to the specified remote TDI Server.

Note: If the Username/Password (in case of custom authentication) or SSL key stores/trust stores (in case of SSL) are not set up correctly, then AMC will be unable to connect to the remote TDI Server and show that server as "Stopped" or "Not running".

AMC and User/Group/Role Management

The Console Administrator (also called superadmin) is the administrator for AMC. This user has complete privilege over AMC and has access to all functions of AMC. Much like the root user of an OS. By default, when AMC is installed for the first time, this is the only user that exists. Default password for "superadmin" user is "secret". It is recommended to change this password on first login. To change the password, go to the **User Preferences->Change Password** panel of AMC. The console administrator's password is stored in a file called the **console_passwd** in the **tdiamc/WEB-INF/classes/security** folder. This file only contains no other information except the Console Administrators login credentials (encrypted).

AMC allows the superadmin user to add users and groups to AMC. By default, AMC comes configured for Cloudscape Database as the storage repository for users and groups. To add users to AMC, login in as superadmin, and go to the Users and Groups section of the navigation area. For instance, if you add a user called "test" with password "sec123", then you can use this user name and password to login into AMC.

Note: The console administrator has no restrictions whatsoever while working with AMC. But if any other user logs in (besides superadmin) then the following functionality is NOT available:

- Manage Console Properties
- Create Config View
- Users and Groups (Add/Edit/View/Manage)

Just like Users can be added to the AMC's database, Groups can also be created by superadmin. Once a group is created, to add users to that group, the console administrator needs to "Edit" that group to be able to Add or Remove members to that group.

AMC and LDAP as an Authentication Store

Instead of using AMC's database as an authentication repository for AMC, users can also use any third party LDAP Server (like IBM Tivoli Directory Server, Active Directory, etc.) as an authentication repository. In this case, whenever a user attempts to login into AMC (except the superadmin), AMC will attempt to construct the user's DN based on the configured search scope for AMC and then using the DN and password provided attempt to bind to the LDAP Server. If the bind succeeds, user will be logged into AMC, if not, then a login failure message will be shown to the user.

Note: AMC will not modify or update the 3rd party LDAP Server repository in any way. This means that AMC treats the LDAP repository only as a lookup for authentication and treats it as read-only. Users can specify appropriate Bind DN for AMC in the AMC LDAP configuration settings – such that the DN has read-only permissions on the LDAP Server. Because of this security restriction, AMC does **not** allow User Addition, User Modification, Password Change, Group Addition and Group Modification via AMC – when AMC is configured for LDAP Server as an authentication store.

Steps needed for configuring AMC to use LDAP as an authentication store:

1. Start AMC.
2. Login as console administrator (superadmin).
3. Go to **Console Administration->Manage Console Properties->General panel**.
4. Set the Authentication Mode dropdown to "**LDAP**" – indicating that the Authentication repository must be treated as LDAP instead of Database (default).
5. Go to **Console Administration->Manage Console Properties->LDAP Properties** panel.
6. Select the "**SSL Enabled**" check box if LDAP Server is running in SSL Mode. In this case you must ensure that appropriate certificates are imported into AMC's trust store and into LDAP Servers trust store. AMC's trust store and key store settings can be viewed, edited from "SSL Settings" panel on the same screen.
7. Enter the LDAP Server hostname or IP address (example: lookup.in.ibm.com)
8. Enter the port on which the LDAP Server is listening (example: 389)
9. Enter the Bind ID and Bind Password with which AMC will bind with the remote TDI Server. AMC uses these credentials to obtain a list of authorized AMC users and groups. This ID can have read-only privilege on the LDAP Server, since AMC does not need any write permission on LDAP Server (example cn=amc,cn=admin,o=ibm,c=us).
10. Enter the suffix DN. All AMC users and groups will be searched for under this suffix (example: o=ibm,c=us).
11. Enter the LDAP user prefix (example: cn)
12. Enter the LDAP user suffix (example: ou=AMCUsers)
13. Enter the LDAP Group Prefix (example: cn)
14. Enter the LDAP Group Suffix (example: ou=AMCGroups)
15. Enter the LDAP User Object class (example: inetOrgPerson)
16. Enter the LDAP Group Object class (example: groupOfNames)
17. Enter the LDAP Group Member (example: member)

18. Enter the LDAP Search timeout – in seconds (example: 120)
19. Click OK.
20. Restart AMC.

With this, AMC will be setup to allow only those users who
- have DN ending with ou=AMCUser, o=ibm, c=us
- have prefix as "cn=",
- have objectclass as "inetOrgPerson"

So, cn=Test, ou=AMCUser, o=ibm, c=us is a valid user; whereas uid=Test, ou=AMCUser, o=ibm, c=us is an invalid user (since it does not have "cn" as prefix). While logging in the user id entered must be only "Test" – and AMC will automatically construct the above DN and attempt to bind with the user supplied password.

Similarly, with the above given example values, AMC will be setup to show only those groups which
- have DN ending with ou=AMCGroups, o=ibm, c=us
- have prefix as "cn="
- have objectclass as "groupOfNames"
- Valid members will be the ones mentioned in the "members" attribute of this objectclass.

Notes:
1. 1. All these properties are stored in the **amc.properties** file.
2. The property – Admin UID, Admin Password, Server Type, LDAP User filter, LDAP Group Filter and LDAP Ignore Case are not being used currently and can be set to any value. They are put there for later use depending on future requirements.

AMC and Role Management

Every user (or group) in AMC can be assigned one of the following roles in AMC for a particular Config View. This role assignment can be done in the **Config Administration->Manage Config View** panel by selecting a particular Config View and clicking on "Configure ACLs" button. The available roles are:
- Read
- Execute
- Admin
- Config Admin

These roles are in increasing order of privilege – indicating that Config Admin is the highest privilege and Read is the lowest. Any functionality that is available to a user with "Read" role for a Config View, will definitely be available to a user with "Execute" privilege on that Config View. Any functionality that is available to a user with "Execute" privilege on a Config View, will be available to a user with "Admin" privilege, and so on.

The following is the meaning of these roles

Read This means that this user can only read the "details" of this Config View – such as what are the ALs inside this view, what are properties inside this view, what is the status of these ALs, etc. This user cannot modify, start, stop, or change any detail of this config view.

Execute
This is essentially a Read user with one extra privilege – the ability to Start and Stop Assembly Lines.

Admin
This user can administer the config view, without being able to modify the config view itself. This

user can do everything that the “Execute” privilege user can do, and additionally he can modify properties, delete logs, configure AM rules, etc for this config view.

Config_Admin
This user can virtually do anything to the config view – including modifying the view itself, modifying the permissions of other users on this view, etc. This is the highest privilege that can be given to a user for a particular config view.

The above roles can be assigned to any Group too. Therefore, if a user “test” and “tdi” are part of the “DBAdmin” group, and the “DBAdmin” group is given “ConfigAdmin” privilege over a config view “SynchDatabase”, then both “test” and “tdi” will automatically get ConfigAdmin privilege over the “SynchDatabase” config view.

Notes:

1. If the “test” user is explicitly given “read” privilege for the same config view, then “Read” will get precedence over the privilege he gets from being part of the “DBAdmin” group. This is done so that “specific” role assignment gets priority over role assignment from groups. This allows people to restrict or give higher access to individuals – without worrying about inherited access from being part of some groups.
2. If the “test” user is part of two Groups – where Group1 has “read” access and Group2 has “admin” access over the same Config View – then in this case the test user will get the higher of the two privilege – in this case being “admin”, unless a specific role is already assigned to “test” for the same config view – in which case the specific role assigned to “test” will be given precedence [point 1 above].

AMC and Passwords

Any password field that is stored in amc.properties file, such as LDAP Bind password, key store password, etc are all encrypted before being written to amc.properties file. Also, AMC never displays any Password fields or protected fields on console. All such fields are masked out.

Users can change their login password (if the backend repository is not LDAP) by going to the **User Preferences->Change Password** panel.

AMC and Encrypted Configs

AMC allows users to load and connect to password protected configs. On the Load Reload panel of AMC, a password text box has been provided – where the users need to enter the password of the config they are attempting to start before clicking “Start”. Similarly, in the AM Configuration Screen – for the Start AssemblyLine action, a password field has been provided where the user can enter the password of the config. Action Manager will pass this password while attempting to start the Config.

Note: AMC cannot detect that the remote config being started is a password protected config. For this reason, if the password is not specified or incorrectly specified, then the user will just see an error message saying – “Unable to start the config”. The user can see the TDI Server logs where an exact message will be provided.

Logging into the console

Open a Web browser and type the following address:

```
http://localhost:<port>/tdiamc
```

Where *port* is 13100 if you installed Embedded WAS Express 6.0.2 using the TDI Installer, otherwise it should be the port where your web server is running.

The IBM Tivoli Directory Integrator Administration and Monitoring Console login page panel is displayed.

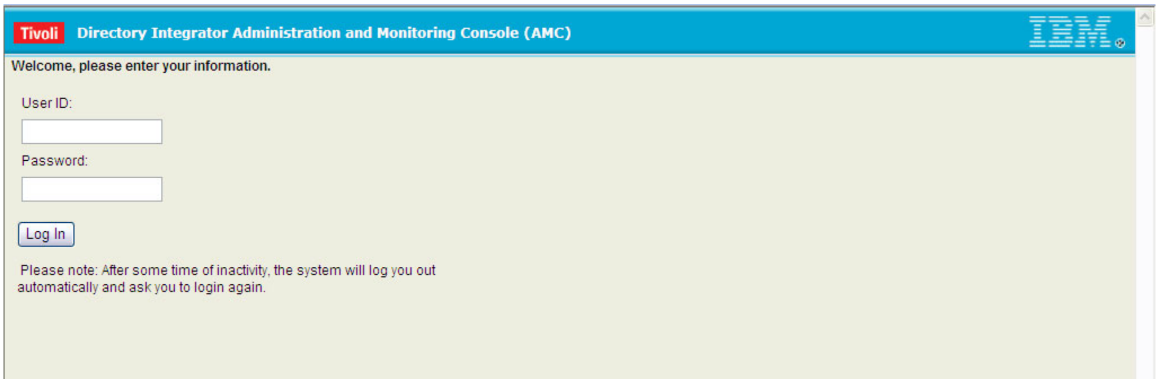
Logging on to the console as the console administrator

The console administrator is a user who can:

- Configure the properties required for the AMC
- Set the authentication mechanism used for AMC logins
- Add new users and configure users' roles

At the IBM Tivoli Directory Integrator Administration and Monitoring Console login page, log in as Console Admin:

1. If the default user name has not been changed, type **superadmin** in the **User ID** field,
2. If the default password has not been changed, type **secret** in the **Password** field.
3. Click **Log In**. The TDI Administration and Monitoring Console is displayed.


Tivoli
Directory Integrator Administration and Monitoring Console (AMC)
IBM
User ID:
Password:
Log In

Welcome, please enter your information.

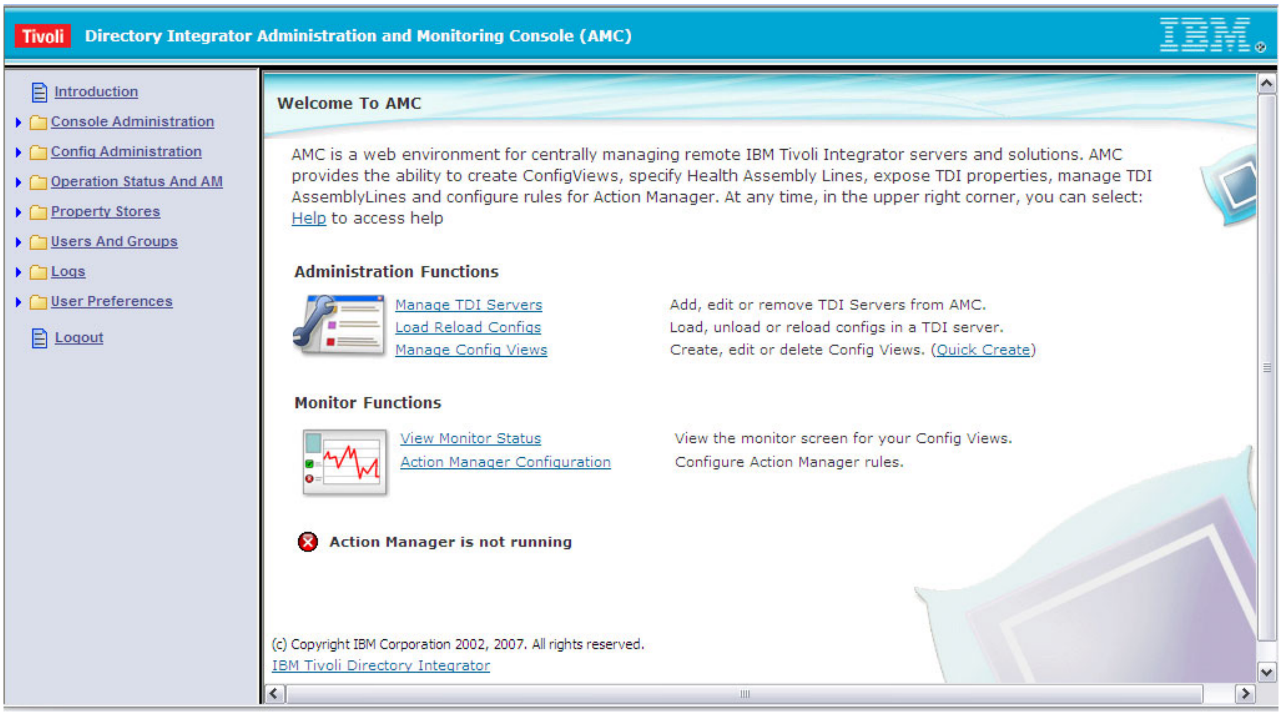
Please note: After some time of inactivity, the system will log you out automatically and ask you to login again.

AMC Console Layout

The IBM Tivoli Directory Integrator Administration and Monitoring Console includes the following components:

Navigation Area
The Navigation area provides a tree view that allows users to navigate through the tasks available to the user in the console. You can open and close folders in the navigation area and select tasks (non-folders) to launch in the Work Area of the console framework.

Work Area
The Work Area contains the necessary information and input fields to complete the task you are currently working on.



Welcome To AMC

AMC is a web environment for centrally managing remote IBM Tivoli Integrator servers and solutions. AMC provides the ability to create ConfigViews, specify Health Assembly Lines, expose TDI properties, manage TDI AssemblyLines and configure rules for Action Manager. At any time, in the upper right corner, you can select: Help to access help

Administration Functions

Manage TDI Servers — Add, edit or remove TDI Servers from AMC.
Load Reload Configs — Load, unload or reload configs in a TDI server.
Manage Config Views — Create, edit or delete Config Views. (Quick Create)

Monitor Functions

View Monitor Status — View the monitor screen for your Config Views.
Action Manager Configuration — Configure Action Manager rules.

Action Manager is not running

(c) Copyright IBM Corporation 2002, 2007. All rights reserved.
IBM Tivoli Directory Integrator

Logging off the console

To log off of the console, click **Logout** in the navigation area.

Using AMC tables

The IBM Tivoli Directory Integrator Administration and Monitoring Console displays certain information, such as lists of attributes and entries, in tables. Tables contain several utilities that allow you to search for, organize and perform actions on these table items.

IBM Tivoli Directory Integrator Administration and Monitoring Console tables provide icons to help you organize and find information in the table. Some icons appear on some tables and not on others, depending on the current task. The following is a comprehensive list of the icons you might encounter:

- Click the **Show Filter Row** icon to display filter rows for every column in the table. See Filtering for more information about filtering.
- Click the **Hide Filter Row** icon to hide filter rows for every column in the table. See “Filtering” on page 140 for more information.
- Click the **Clear all filters** icon clear all filters set for the table. See “Filtering” on page 140 for more information.
- Click the **Edit sort** icon to sort the information in the table. See “Sorting” on page 139 for more information.
- Click the **Clear all sorts** icon to clear all sorts set for the table. See “Sorting” on page 139 for more information.
- Click the **Collapse table** icon to hide the table data.
- Click the **Expand table** icon to display the table data.
- Click the **Select all** icon to select all items in the table.
- Click the **Deselect all** icon to deselect all selected items in the table.

Select action drop-down menu

The **Select action** drop-down menu contains a comprehensive list of all available actions for a selected table. For example, instead of using the icons to display and hide sorts and filters, you can use the **Select action** drop-down menu. You can also use the **Select action** drop-down menu to perform operations on the table contents; for example, on the **Manage attributes** panel, actions such as **View**, **Add**, **Edit**, **Copy** and **Delete** appear not only as buttons on the toolbar, but also in the **Select action** drop-down menu. If the table supports it, you can also display or hide the **Show find** toolbar using the **Select action** drop-down menu. See Finding for more information on finding table items.

To perform an action using the Select action menu:

1. If necessary, select an item from the table.
2. Click the **Select action** drop-down menu.
3. Select the action you want to perform; for example **Shutdown server**.
4. Click **Go**.

Paging

To view different table pages, use the navigation controls at the bottom of the table. You can enter a specific page number into the navigation field and click **Go** to display a certain page. You can also use the **Next** and **Previous** arrows to move from page to page.

Sorting

To change the way items in a table are sorted:

1. Do one of the following:
   - Click the **Edit sort** icon on the table.
   - Click the **Select action** drop-down menu, select **Edit** sort and click **Go**.

   A sorting drop-down menu appears for every column in the table.
2. From the first sort drop-down menu, select the column on which you'd like to sort. Do the same for any of the other sortable columns on which you'd like to sort.
3. Select whether to sort in ascending or descending order by selecting **Ascending/ Descending** from the drop-down menu. Ascending is the default sort order. You can also sort using column headers. On every column is a small arrow. An arrow pointing up means that column is sorted in ascending order. An arrow pointing down means that column is sorted in descending order. To change the sort order, simply click on the column header.
4. When you are ready to sort, click **Sort**.

To clear all the sorts, click the **Clear all sorts** icon.

Finding

To find a specific item or items in a table:

Note: The Show find toolbar option is available on some tables and not on others, depending on the current task.

1. Select **Show find toolbar** from the **Select action** drop-down menu and click **Go**.
2. Enter your search criteria in the **Search for** field.
3. If desired, select a condition upon which to search from the **Conditions** drop-down menu. The options for this menu are:
   - **Contains**
   - **Starts with**
   - **Ends with**
   - **Exact match**

4. Select the column upon which you want to base the search from the **Column** drop-down menu.
5. Select whether to display results in descending or ascending order from the **Direction** drop-down menu. Select **Down** to display results in descending order. Select **Up** to display results in ascending order.
6. Select the **Match case** check box if you want search results to match the upper and lower case criteria in the **Search for** field.
7. When you have entered the desired criteria, click **Find** to search for the attributes.

Filtering

To filter items in a table, do the following:

1. Do one of the following:
   - Click the **Show filter** row icon. Click the **Select action** drop-down menu, select **Show filter** row and click **Go**.
2. Filter buttons appear above each column. Click **Filter** above the column on which you want to filter.
3. Select one of the following conditions from the **Conditions** drop-down menu:
   - Contains
   - Starts with
   - Ends with
4. Enter the text you want to filter on in the field; for example, if you selected **Starts with**, you might enter **C**.
5. If you want to match case (upper case text or lower case text) select the **Match case** check box.
6. When you are ready to filter the attributes, click **OK**.
7. Repeat the above steps 2-6 for every column on which you want to filter.

To clear all the filters, click the **Clear all filters** icon.

To hide the filter rows, click the **Show filter** icon again.

Console Administration

If you have not done so already, expand the Console Administration category in the navigation area of the Administration and Monitoring Console.

Do one of the following:

- To add and edit IBM Tivoli Directory Integrator servers, click **Manage TDI Servers** .
- To set general, LDAP, SSL and miscellaneous console settings, click **Manage Console Properties** .

Manage TDI Servers

This panel allows you to view the registered server. Additionally, the Console Administrator can add, edit, delete and shut down IBM Tivoli Directory Integrator servers from this panel.

When AMC is started , it automatically has a local TDI server on port 1099 registered. Therefore, in the Manage TDI Server panel, one entry in under LOCAL TDI SERVER is already present with its state depicted as Running or Stopped depending on its status

You can choose the operations you want to perform from the tool bar at the top of the table or using the Select action drop-down menu, such as:

Add click the Add button on the toolbar.

Delete Select the radio button next to the server you want to delete and click the Delete button on the toolbar.

Edit Select the server you want to edit and click the Edit button on the toolbar.

Shutdown server
Select the server you want to shut down and click the Shutdown Server button on the toolbar.

Add a server

This panel allows you to add an IBM Tivoli Directory Integrator server to the Administration and Monitoring Console (AMC). Once you have added a IBM Tivoli Directory Integrator server to the AMC, you can then utilize features on other AMC panels to add Config Views to the TDI server and to create and define views for the Config Views associated with the IBM Tivoli Directory Integrator server.

To add a new TDI server:

1. Enter a name for the IBM Tivoli Directory Integrator server in the **Name** field.
2. Enter the host name or IP address of the computer on which the IBM Tivoli Directory Integrator is running in the **Hostname** field.
3. Enter the port number on which the IBM Tivoli Directory Integrator server is configured to run.
4. Select the desired authentication mode. If you selected the Custom or LDAP authentication authentication method, enter the username and password to be used for authentication.
5. Click **OK**.

Edit a server

This panel allows you to edit an existing IBM Tivoli Directory Integrator server. To edit an existing server:

1. Enter the host name or IP address of the computer on which the IBM Tivoli Directory Integrator server is running in the **Hostname** field.
2. Enter the port number on which the IBM Tivoli Directory Integrator server is configured to run.
3. Select the desired authentication mode. If you selected the Custom or LDAP authentication authentication method, enter the username and password to be used for authentication.
4. Click Cancel to exit the panel without making any changes, or click **OK** to save the changes.

Manage Console Properties

From these panels, you can set General, LDAP, SSL and Miscellaneous console settings.

General

This panel allows you set general properties such as refresh rates and session timeouts for the Administration and Monitoring Console. From this panel you can:

- Set the ″Monitor Status″ panel as the default panel that appears after logging on to the Administration and Monitoring Console.
- Set the monitor screen refresh rate in minutes
- Set the frequency (in days) at which Action Manager logs are rotated.
- Set the Administration and Monitoring Console session timeout in minutes.

  Note: The session timeout parameter is set when a user logs into AMC. Therefore, this parameter will only take effect from next user login.

LDAP

The Administration and Monitoring Console provides a way for you to authenticate users to an LDAP backend.. If LDAP authentication is enabled, you must configure the properties of the LDAP server to which you want to authenticate.

SSL

This panel allows you to set up the console so that it can communicate with other directory servers using the Secure Sockets Layer (SSL) encryption, if necessary.

The default values for Keystore and Truststore are set to testadmin.jks. This testadmin.jks file is present in the `<TDI_Install_Dir>\AppServer\profiles\amcprofile\installedApps\DefaultNode\tdiamc.war.ear\tdiamc.war\` directory. The keystore and truststore password are set to *"administrator"* by default.

Miscellaneous

JDBC properties are used to define the connections settings to the Cloudscape database. The Cloudscape database is used to store the Administration and Monitoring Console's configuration information, connection details, and Action Manager rules and results.

From this panel you can:

- Enter the JDBC URL in the **JDBC URL** field.
- Enter the desired user name in the **Username** field.
- Enter the password for the user in the **Password** field.
- Enter the JDBC driver class name in the **JDBC driver** field.

Config Administration

If you have not done so already, expand the Config Administration category in the navigation area of the Administration and Monitoring Console.

Do one of the following:

- To create a Config View, click **Create Config Views**.
- To view, add, delete and edit Config Views, click **Manage Config Views**.
- To load or reload a Config, click **Load/Reload Config**.
- To view a Config View's details, click **Config Report**.

Create a Config View

The purpose of a Config View is to give users access to information in the configuration file without granting them the ability to edit the configuration file directly. Administrators can use a Config View to filter a configuration file for specific information so that only certain information within the configuration file is displayed. You can create multiple Config Views for each Config, with each view exposing different information contained in the configuration file.

If you have not done so already, expand Config Administration in the Administration and Monitoring Console navigation area. Click **Create Config Views** in the expanded list.

To create a Config View:

1. Enter view details:
   a. Enter a name for the Config View in the **Name** field.
   b. Enter a description of the Config View in the **Description** field.
   c. Click **Next**.
2. Select the server and configuration file you want to use to create a Config View:
   - From the **Server** drop-down menu, select the IBM Tivoli Directory Integrator server containing the configuration file you want to use to create a Config View. This menu will be empty if no IBM Tivoli Directory Integrator servers have been added to the Administration and Monitoring Console.
   - Select the configuration file you want to use to create a Config View from the **Configs** drop-down menu. The menu contains all currently loaded Configs.
   - Click **Next**.
3. Effective with TDI 6.1.1, you can use **Quick Create** options to create a config view. Quick Create options include:

   Create config view from published solution.

Creates the config view from the published solution as specified in the TDI Configuration Editor (CE). This option requires that your active configuration instance have a published solution associated with it, and also requires a TDI 6.1.1 server.

Create config view with all AssemblyLines exposed.
Creates a config view with all AssemblyLines from the config instance exposed, and no properties and no Health AL defined. Use this option for a quick start (useful for development purposes). Available for TDI 6.0 and later servers.

Create config view with all AssemblyLines exposed and all properties.
Creates a config view with all AssemblyLines from the config instance exposed, and all properties and no Health AL defined. This option does not expose the Java properties. Available for TDI 6.1 and later servers. Use this option for a quick start (useful for development purposes).

4. Select the AssemblyLines you want to associate with the selected Config View and click **Next**.
5. Select the configuration properties you want to make available for viewing and/or modification.

   Note: Depending on what property store is exposed for the selected Config file you can select Global, Solution, Java, System, Password or User properties.

   a. Select the desired property store from **Select property store** drop-down menu.
   b. Select the desired property or properties you want to add to the Config View from the **Select Properties** table
   c. Click **Next**.
6. Select the Health AssemblyLine you want to monitor from the **Select Health AssemblyLine** table. If you do not wish to monitor heartbeats, select the **None** check box.
7. Click **Finish**.

Manage Config Views

To view, add, delete and edit Config Views, click Manage Config Views:

- To add a Config View, click the **Add** button on the toolbar.
- To modify an existing Config View, select the property you want to edit and click the **Edit** button on the toolbar.
- To configure ACLs for a Config View, select the Config View for which you want to configure ACLs and click the **Configure ACLs** button on the toolbar.
- To delete an existing Config View, select the property you want to delete and click the **Delete** button on the toolbar.

Configure ACLs

From this panel you can set the Access Control Lists (ACLs) for a user and associate that user with a specific Config View.

- To configure a user or users, select the user or users you want to configure and click the **Configure Users** button on the toolbar.
  1. Select the user you want assign a role to from the **User ID** drop-down menu.
  2. Select the radio button next to the role or roles you want to assign the selected user:
     - **Read** - Allows the user to read Config View details like ALs, Tombstones, logs, properties belonging to the Config, etc.
     - **Execute** - Allows the user to read and start/stop AssemblyLines
     - **Admin** - Grants the user Reader and Execute roles. This role also allows users to delete logs and tombstones.
     - **Config Admin** - Grants the user the ability to start and stop a Config, modify the Config View, and assign and modify ACLs for other users.
  3. Click **Apply**.

- To configure a group, select the property you want to edit from the table and click the **Edit** button on the toolbar.
  - Select the group you want assign a role to from the User ID drop-down menu.
  - Select the radio button next to the role or roles you want to assign the selected group:
    - **Read** - Allows the group's members to read Config View details like ALs, Tombstones, logs, properties belonging to the Config, etc.
    - **Execute** - Allows the group to read and start/stop AssemblyLines
    - **Admin** - Grants the group Reader and Execute roles. This role also allows the group to delete logs and tombstones.
  - **Config Admin** - Grants the group the ability to start and stop a Config, modify the Config View, and assign and modify ACLs for other groups.
- To remove an existing user or group, select the user or group from the table and click **Remove**.

When you are finished making changes, click **Apply**.

Load/Reload Configurations

This panel displays loaded Configs and the Configs in the configs folder of the remote IBM Tivoli Directory Integrator server. When AMC is connected to a TDI 6.1 server the Load/Reload configuration screen will show a listing of all files in the remote config folder (irrespective of whether the files are valid TDI config files or not). You should perform Load operations on valid TDI Config files only, otherwise an error message is displayed in AMC.

From this panel you can load, unload and reload a Config onto a server.

Note: You must have superadmin or config admin privileges to perform these actions.

- To load a configuration, select the configuration you want to load and click **Load**.
- To unload a configuration, select the configuration you want to unload and click **Unload**.

  Note: Loading a server does not automatically start the AssemblyLines associated with the selected Config. Only those AssemblyLines designated as AutoStart will start upon loading.
- To reload a Config, select the configuration you want to reload and do one of the following and click **Reload**
- To refresh a Config, select the configuration you want to reload and do one of the following and click **Refresh**

Click **Close** when you are finished making changes.

Config Report

This panel displays the Config Views for which you have at least Reader access.

- To view a Config View select the desired Config View from the **Select Config View** drop-down menu and click **Show Report**.
- To view the AssemblyLine's Components, select the radio button next to the AssemblyLine with Connectors you want to view and click **View Components**.
- To start an AssemblyLine select the radio button next to the AssemblyLine you want to start and click **Start AL**.
- To stop an AssemblyLine, select the radio button next to the AssemblyLine you want to stop and click **Stop AL** .

.

Operation Status and AM

If you have not done so already, expand the **Operation Status** category in the navigation area of the Administration and Monitoring Console.

Do one of the following:

- To view information about each preferred Config View, such as AM Status, Health Check Result and Health Check Status, click **“Monitor Status”**.
- To add, edit or delete Action Manager configuration rules, click **“AM Configuration” on page 147**.

Monitor Status

This panel displays the views selected on the ″Preferred Views″ panel accessed from **User Preferences** in the Administration and Monitoring Console navigation area. It displays high level information about each preferred Config View, such as:

AM Status
Displays the status of the Action Manager rules for the selected Config View: A blue exclamation mark indicates that no Action Manager rules have been triggered recently. An yellow triangle containing an exclamation mark indicates that an Action Manager rule has been triggered recently.

Health Check Result
Displays the health check result obtained from the healthAL.result final work entry attribute in the Config View’s Health AssemblyLine. This value is displayed as text.

Health Check Status
Displays the health check status obtained from the healthAL.status attribute in the Config View’s Health AssemblyLine.

Additionally, if you have designated a .gif file with the same name as the returned status value in the Administration and Monitoring Console’s `resources/amc_images/healthAL` directory, the .gif image will also be displayed in this column. For example, if the healthAL.result is returned as ″Error″, and you have created an ″Error.gif″ in the above mentioned directory, the Error.gif image displays in the table column.

From this panel you can:

- View Config View details - To view the details of a specific Config View, select the desired Config View and click **Config View Details**
- View TDI Server Information - To view the details of the server to which the Config View belongs, click **TDI Server Information**.
- Show Preferred Config Views - Click **Show Preferred Config Views** to view preferred Config Views. This button is visible only if Preferred Config Views are defined. You can define preferred Config Views on the ″Preferred Config Views″ panel under **User Preferences**.

Config View Details

This panel contains two tables. The top table displays the AssemblyLines associated with the selected Config View and the status of each Config View. The bottom table displays information about recently triggered Action Manager rules.

When you are through making changes, click **Close**.

Config View Details Table:

Columns: The **Config View Details** table contains the following columns:

Select Select the radio button next to the AssemblyLine on which you want to perform an action.

AssemblyLines
Displays the name of the AssemblyLine.

Status Displays the AssemblyLine's status; for example, **Running** or **Stopped**.

Start Time

AssemblyLine is running
Start Time is when the running AL started. Start Time is based on the running AL.

AssemblyLine is stopped
The time when the last run of the AL started. Start Time is based on the most recent tombstone entry for the AL. (Available only with TDI 6.1.1 servers).

Last Stop Time
The time when the last run of the AL terminated. Stop Time is based on the most recent tombstone entry for the AL. (Available only with TDI 6.1.1 servers).

Statistics
Displays the current statistics of the running AssemblyLine.

Actions: You can choose the operations you want to perform from the tool bar at the top of the table or using the **Select action** drop-down menu, such as:

- View AL Tombstones - Select the AssemblyLine you want to view and click the **View AL Tombstones** button
- View AL Logs - Select the AssemblyLine you want to view and do one of the following:
  - Click the **View AL Logs** button on the toolbar.
  - Select **View AL Logs** from the **Select action** drop-down menu and click **Go**.
- Manage Properties - Select the radio button next to the AssemblyLine with properties you want to manage and Click the **Manage Properties** button on the toolbar.
- Start AL -
  1. Select the AssemblyLine you want to start
  2. Click the **View pop-up** button
  3. Click **Start AL**.
- Stop AL - Select the AssemblyLine you want to stop and do one of the following:
  1. Select the AssemblyLine you want to stop
  2. Click the **View pop-up** button
  3. Click **Stop AL**.

View Tombstones: If you have tombstones enabled on the remote IBM Tivoli Directory Integrator server, the Administration and Monitoring Console can display the tombstone entries for terminated AssemblyLines. This panel displays useful information about tombstone entries, such as when the entry was changed to the tombstone state.

View AL Logs: To view the list of log files for the selected AssemblyLine, click the radio button next to the log you want to view and click **View AL Logs**.

Note: In order to view an AssemblyLine log in the Administration and Monitoring Console, the AssemblyLine must log using the SystemLog logger.

AM results table: When a rule set in the Action Manager is triggered, information about the violation is logged, such as the source of the violation, a description of the error and the time at which the violation occurred. These details are displayed in the **AM Results** table.

The following sections contain information about the **AM Results** table columns and how to perform operations on AM Results.

Columns: The **AM Results** table contains the following columns:

Select Select the radio button next to the message on which you want to perform an action.

Source
Displays the name of the Action Manager rule that was triggered.

Severity
Displays the severity of the message.

Message
Displays the message associated with the AM action.

Description

Displays additional information about the message.

Timestamp

Displays the time at which the Action Manager rule was triggered and the message was generated.

Actions: Select the result or results you want to delete and click **Delete**.

TDI Server Information

This panel displays the IBM Tivoli Directory Integrator server information of the server to which the currently selected Config View belongs. The information on this panel is read-only, although administrators have the capability to shut the server down from this panel.

Show Preferred Config Views

Preferred Config Views are the default Config Views that are displayed on ″Monitor Status″ panel.

AM Configuration

This panel allows you to add, delete or modify AM rules, triggers and actions to be performed as a result of rules execution and triggering conditions.

Add/Edit configuration rules

Using the settings on this panel you can create an Action Manager rule (or modify an existing one) for the current Config View.

A rule consists of two parts:

- The condition under which the rule is to be invoked, called a ″trigger.″

  Some examples of triggers are Server API failure, AssemblyLine failure, or failure of the AssemblyLine to run at the specified intervals.
- A set of alternate actions to be performed when the trigger is encountered.

Configuration rules settings: This panel is concerned with the first part of the rule: defining triggers. From the panel you can select a name, description, and trigger type.

Name
Enter a name for the rule. If you are adding a rule, this field is required.

Description
Enter an optional description of the rule.

Trigger type
The trigger type defines the conditions under which a rule is invoked. From the drop-down menu, select a trigger type:

No trigger
Rule has no triggering condition.

On AL termination
Rule is triggered when the specified AssemblyLine is terminated.

Time since last execution
Rule is triggered when the specified AssemblyLine has not run for the determined period of time.

On Query AL result
Rule is triggered when the last ″work″ entry of the specified AssemblyLine contains an attribute matching a given condition and value.

On server API failure
Rule is triggered when the Action Manager is unable to connect to the remote server using the Server API. This rule is triggered only once. The rule resets when it detects that it can reconnect to the server using the Server API.

On received Event
Rule is triggered when the Action Manager receives an event that meets the criteria specified in the Event type, Event Source and Event Data fields.

On Property
Rule is triggered when the specified property meets the determined Property name, Condition and Value specifications.

Configure trigger: Each trigger type has a different selection of settings. If you do not see some of the fields listed below on your panel, it is because the trigger type you currently have selected does not support them.

Source
Enter the source you want to monitor.

Data Enter the data you want to monitor.

Property name
From the drop-down menu, select the property name you want to monitor.

Condition
Select the condition you want to use to compare the property and value. Possible options are:

- equals
- not equals
- greater than
- less than

Value Enter the value you want to monitor.

Configured actions: From this table you can add, delete and edit actions. You can also move actions up and down in the table:

- To add an action, click the **Add** button.
- To delete an action, select the action you want to delete and click the **Delete** button.
- To edit an action, select the action you want to edit and click the **Edit** button.
- To move an action up one position in the table, select the action you want to move and click **Move Up**.
- To move an action down one position in the table, select the action you want to move and click **Move Down**.

Add/Edit Action

When a rule is triggered, the Action Manager executes the actions associated with the rule. This panel allows you to specify or modify the actions you want Action Manager to take when the rule is triggered.

From the drop-down menu, select an action type, and configure it. Click **OK** when you are finished.

Start AssemblyLine

This action starts an AssemblyLine. If you select this action, you must specify the name of the AssemblyLine you want to start and its associated Config (and possibly the Config's password).

Server This is a drop-down list of configured Servers. LocalServer means the Server on the machine AM is executing.

Select from remote config folder
Checkbox; if enabled, queries the remote Server for available Config files. The Config files displayed are those present in the folder whose path is specified for the api.config.folder property in the `global.properties` file.

Config name
Enter the Config to which the AssemblyLine in the AssemblyLine field belongs. If **Select from remote config folder** is checked, you will be presented with a list of available Config files on the remote Server, if unchecked, you must fill in the name of a locally-available Config file.

This field is required.

Config password
If required, enter the Config password for the selected Config file. This field is applicable only if the config is password protected.

AssemblyLine
Enter the name of the AssemblyLine to start.

Configure AL Operation
This hyperlink launches the 'Select Operation' dialog. If the AssemblyLine has been defined with one or more custom Operations, this dialog enables you to select such an Operation. Subsequently, you will be prompted for the AL's Initialization attributes and Operation attributes for this Operation. This label is shown only for TDI 6.1 and TDI 6.1.1 servers if configured and is not applicable for TDI 6.0.

Stop AssemblyLine

This action stops an AssemblyLine. If you select this action, you must specify the name of AssemblyLine you want to stop and its associated Config.

Server This is a drop-down list of configured Servers. LocalServer means the Server on the machine AM is executing.

Select from remote config folder
Checkbox; if enabled, queries the remote Server for available Config files.

Config name
Enter the Config to which the AssemblyLine in the AssemblyLine field belongs. If **Select from remote config folder** is checked, you will be presented with a list of available Config files on the remote Server, if unchecked, you must fill in the name of a locally-available Config file.

This field is required.

AssemblyLine
Enter the name of the AssemblyLine to stop.

Enable/Disable AM Rule

Select the Enable/Disable AM Rule to enable or disable an Action Manager rule.

Rule name
Select the name you want to enable or disable from the drop-down menu.

State Select the desired state from the drop-down menu. If you want to enable the rule in the **Rule name** field, select **Enabled**. If you want to disable the rule, the select **Disable**.

Execute AM Rule

This action causes the Action Manager to execute the specified rule. Action Manager then executes the actions associated with the specified rule. The trigger condition associated with the specified rule is not required to be satisfied.

Rule name

Select the name of the rule you want the Execute AM rule to execute from the drop-down menu.

Notify Event

This action causes the Action Manager to send an event with the specified details to the IBM Tivoli Directory Integrator server associated with the current Config View. To add this action to the rule, select the **Notify event** check box. If you select this action, you must specify an event type.

Event type

Enter an event type. This field is required.

Source

Enter a source for the event type.

Data Enter data for the event type.

Modify property

This action causes the Action Manager to modify a property based on a specific operation and value. If you select this action, you must also select a value.

Property name

Select the property you want to modify from the drop-down menu.

Operation

From the drop-down menu, select the operation you want to use to modify the property. Possible options are:

- Set
- Increment
- Decrement

Value Enter the desired value. This is a required field.

Copy TDI property value

This action causes the Action Manager to copy the value of the source property to the destination property.

From property

From the drop-down menu, select the property you want to copy from.

To property

From the drop-down menu, select the property you want to copy to.

Write to log

This action creates a log of the Action Manager rules that have been invoked, according to the specified severity, message and description. This log can be viewed under **Monitor Status**, on the ″Config View Details″ panel in the **AM results** table. Having at least one log action for every rule is recommended. If you select this action, you must enter a message in the **Message** field.

Severity

Select the desired severity from the drop-down menu. Possible options are:

- Severe
- Warning
- Info
- Fine

Message
Enter the desired message.

Description
Optionally, enter a description.

View Current Configuration

To view the current Action Manager configuration for the selected AssemblyLine, click **View Current Configuration**. The table lists all the defined rules, triggers and actions associated with the Config View. When you are done viewing, click **Close**. Only those rules which are in Enabled state are listed here .

Manage Property Stores

If you have not done so already, expand the **Property Stores** category in the navigation area of the Administration and Monitoring Console. To add or edit Java, Solutions, Global, System, User Property and Password Store properties, click **Manage Property Stores**.

When you are done entering the desired the property values, click **OK** to save your changes.

The order in which these Property Stores are listed is significant. The Property Stores are evaluated from top to bottom, but the last definition of a given Property is the one that will be used. By default, the system is set up such that properties defined in a solution-specific properties file called `solution.properties` (residing in the Solution Directory) override corresponding ones in the system-wide `global.properties` file.

Note: Certain System Properties and Java Properties are read-only; they are shown in the respective Property Stores but attempting to modify them will have no effect.

Select Config View

This panel allows you select a Config View. The menu contains only those Config Views for which you have any type of access rights amongst *Read*, *Execute*, *Config_Admin* or *Admin*. Nothing is displayed if you do not have access to any of the config views being created. Once you have selected a view, click **Set**.

After you select a Config View, you can manage properties by clicking on the other property tabs, such as **Solution Properties** and **Global Properties**.

Solution Properties

This panel allows you to add, edit and delete properties in the Solution Properties list.

Global Properties

This panel allows you to add, edit and delete Global Properties.

Java Properties

This panel allows you to add, edit and delete Java properties.

System Properties

This panel allows you to add, edit and delete System properties.

Password Store

This panel allows you to add, edit and delete properties in the Password Store.

User Property Store

This panel allows you to add, edit and delete properties in the Use Property Stores list.

The Property Stores drop-down menu contains a list of property stores configured by the user. Global, Solution, Java and Password Stores properties are not included. From the drop-down menu select the property store whose associated properties you wish to view, add, edit or delete.

Users and Groups

If you have not done so already, expand the Users and Groups category in the navigation area of the Administration and Monitoring Console. Do one of the following:

- To add a user, click **Add Users**.
- To view, add, delete and edit users, click **Manage Users**.
- To add a group, click **Add Group**.
- To view, add, delete and edit groups, click **Manage Groups**.

Add users

From this panel you can add a user to the Administration and Monitoring Console.

To add a user:

1. Enter a user ID in the **User ID** field.
2. Enter a password for the user in the Password field and confirm it in the **Confirm password** field.
3. Enter a description of the user in the **Description** field.
4. Enter the user's email address in the **Email address** field.
5. Click **OK** to add the user, or click **Cancel** to return to the ″Manage Users″ panel without making any changes.

Manage Users

This panel will show the users currently added to AMC, and allows you to add, edit and delete users.

Keep in mind that if you delete a user, the deleted user will be removed from all config views the user is associated with.

The User ID field in the add, edit and delete user panels is not editable. Password fields are shown empty, and a password cannot be retrieved but can be changed by specifying (and confirming) a new password value.

Add Group

This panel allows you to create a group. After entering a name and description, click **OK** to add the group.

Manage Group

This panel allows you to add, edit and delete groups. It also enables you to manage group memberships.

Keep in mind that if you delete a group, it will be removed from all config views the group is associated with.

Edit Group

The Edit Group panel lets you the change the name of a Group, and it also enables you to manage group membership of the group.

Add... Adds a defined User to this Group. Clicking the Add... button will bring up a list of defined Users, which can then be selected and thus added to the Group.

Remove

Removes a selected User from the Group. The User identity itself is not affected (except his membership in this Group).

Cleanup Logs

If you have not done so already, expand the **Logs** category in the navigation area of the Administration and Monitoring Console. To delete log files for all AssemblyLines, for a particular AssemblyLine, or to delete by date, click **Cleanup Logs**.

This panel allows you remove older log files associated with specific AssemblyLines. You can choose to delete log files for all AssemblyLines, or for a particular AssemblyLine. You can also specify logs to delete by date. To clean up logs:

1. Select the Config View with logs you want to clean up from the **Config View** drop-down menu.
2. Do one of the following:
   - Select the **All assembly lines** radio button to delete the logs of all AssemblyLines within the selected Config View.
   - Select the **Specific assembly line** radio button to delete only those logs associated with a specific AssemblyLine.
3. If your selected **Specific assembly line**, select the AssemblyLine with logs you want to delete from the drop-down menu.
4. Do one of the following:
   - To delete logs older than a certain date, select the **Older than** radio button. Enter the desired date in the date field; its format is locale-dependent. You can also use the Calender button, which lets you specify a date by choosing from a calender.

     All logs older than the date specified will be deleted.
   - To preserve more recent logs, select the **Keep most recent** radio button. Enter the number of recent logs you want to save. For example, if you enter the number 10, the 10 most recent logs will be saved.
5. Click **Cleanup** to remove the specified logs or click **Cancel** to exit this panel without making any changes.

User Preferences

If you have not done so already, expand the User Preferences category in the navigation area of the Administration and Monitoring Console. Do one of the following:

- To change your user password, click **Change Password**.
- To designate preferred Config Views, click **Preferred Config Views**

Change Password

This panel allows you to change your login password.

To change your password:

1. Enter your current password in the **Current password** field.
2. Enter your new password in the **New password** field.
3. Confirm your new password by entering it in the **Confirm password** field.
4. Click **OK** to save your changes.

Preferred Config Views

This panel displays a table containing a list of Config Views.

Select those Config Views you want to designate as preferred. When you have selected the desired Config Views from the table, click **OK**.

Preferred Config Views are the default Config Views that are displayed on ″Monitor Status″ panel.

Chapter 12. Tombstone Manager

Introduction

TDI 6.1.1 can keep track of configurations or AssemblyLines that have terminated. This way, you can tell when your AssemblyLines last ran, without going into the log of each one.

This is accomplished by TDI's ″Tombstone Manager″ that creates ″tombstones″ for each AssemblyLine and configuration as they terminate, that contain exit status and other information that later can be requested through the Server API. This also enables:

- A status screen in AMC which displays status of an entire TDI configuration
- Functionality within Action Manager to ensure repeated runs of AssemblyLines, for example every 24 hours
- Provision of status information to Server API clients about AssemblyLines that they run asynchronously.

The Tombstone Manager API is documented in the JavaDocs; look for class `com.ibm.di.api.Tombstone` .

Configuring Tombstones

The creation of Tombstones for AssemblyLines, EventHandlers and Config Instances is configured by means of checkboxes in a number of screens in the Config Editor (CE), as well as a number of options in the `global.properties` or `solution.properties` files.

Once configured, your Config will contain the following switches:

At the configuration level:

- Config switch: specifies whether tombstones will be created or not for the Config Instance itself
- All AssemblyLines switch: specifies whether tombstones will be created for all AssemblyLines from this configuration
- All EventHandlers switch: specifies whether tombstones will be created for all EventHandlers from this configuration

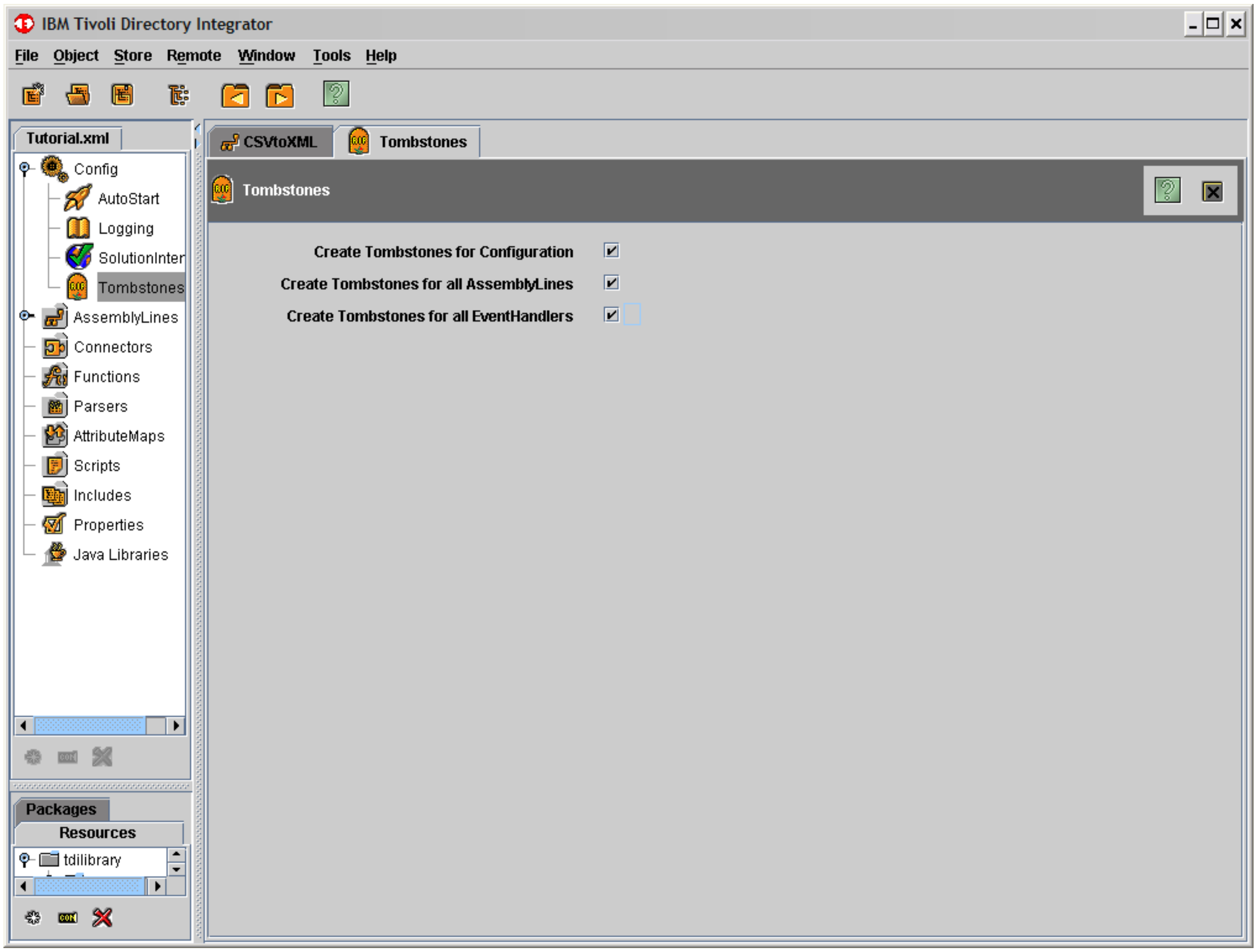
At the AssemblyLine level:

A switch that specifies whether tombstones will be created for that particular AssemblyLine. This switch is only taken into account when the ″All AssemblyLines switch″ at the configuration level is switched off.

There are no individual switches for EventHandlers. All switches are turned off by default.

Config Editor Configuration screen

You can configure Tombstones for this Config Instance by checking the appropriate options in this screen. These switches act as master switches for their respective categories: Configuration, AssemblyLines and EventHandlers.



Create Tombstones for Configuration ☑

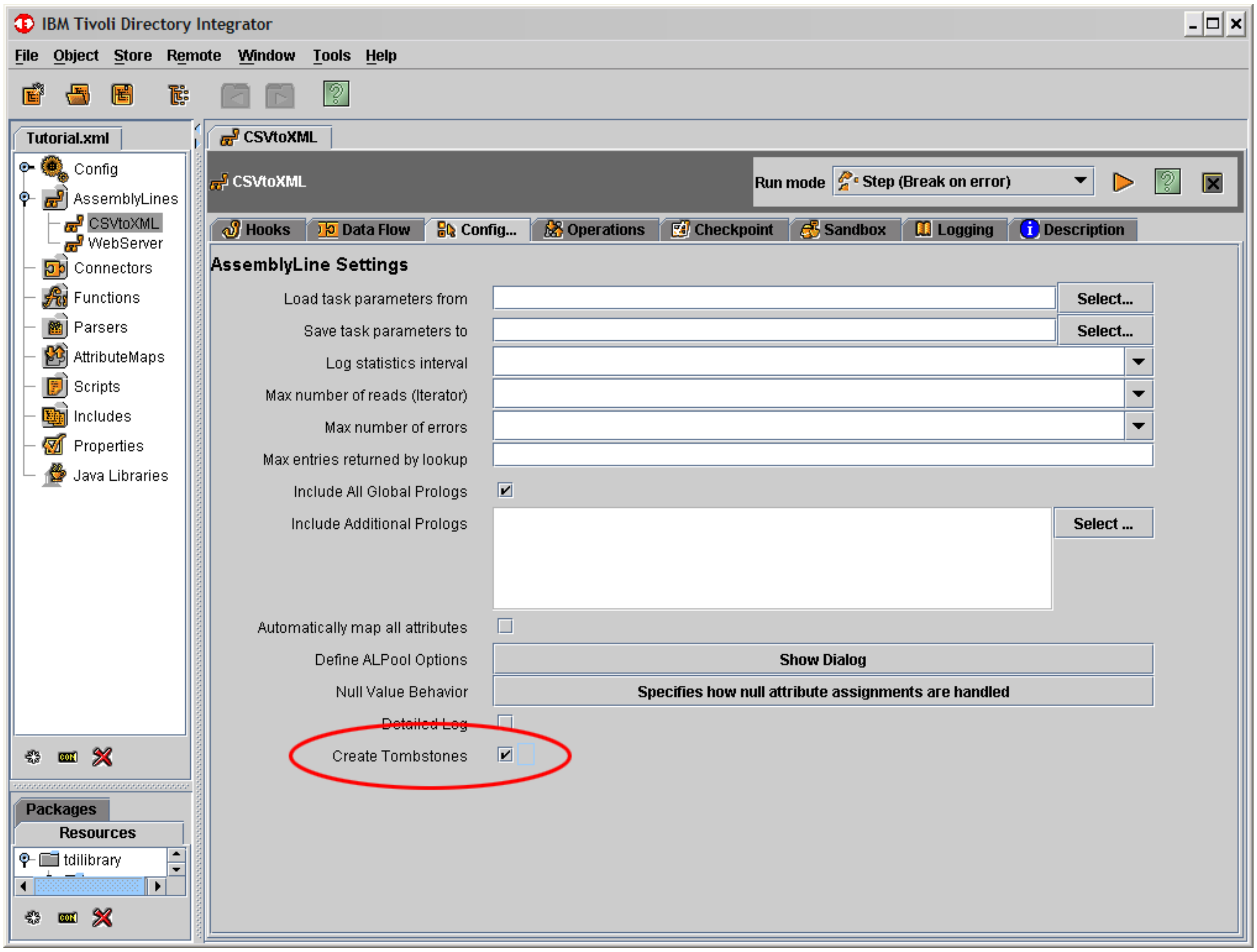
Create Tombstones for all AssemblyLines ☑

Create Tombstones for all EventHandlers ☑

Tombstone creation for Configuration/All AssemblyLines/All EventHandlers is turned on when the corresponding check box is checked.

AssemblyLine Configuration screen

When AssemblyLine Tombstones are disabled using the Configuration option shown above, Tombstone generation can still be enabled individually per AssemblyLine by using the appropriate option in the AssemblyLine configuration screen: **Create Tombstones**, as shown in the screen below:



AssemblyLine Settings

Load task parameters from Select...

Save task parameters to Select...

Log statistics interval

Max number of reads (Iterator)

Max number of errors

Max entries returned by lookup

Include All Global Prologs ☑

Include Additional Prologs Select ...

Automatically map all attributes □

Define ALPool Options Show Dialog

Null Value Behavior Specifies how null attribute assignments are handled

Detailed Log □

Create Tombstones ☑

Checking **Create Tombstones** will cause Tombstones to be generated for this AssemblyLine when it is run, even when the master switch for AssemblyLines is disabled.

Note: There is no Tombstone option in the EventHandler configuration screen. EventHandler Tombstones are configured using the master switch only, and are thus generated either for all EventHandlers activated in a Config, or for none at all.

A sample Tombstone record could look like this:

Table 8.

Field Name	Value
Component Type ID	1
Event Type ID	0
StartTime	11.11.2005 11:11:54
TombstoneCreateTime	11.11.2005 17:22:45
Component Name	"ActiveDirectoryChangeLogSynchronizer"
Configuration	"C:\TDI_SOL_DIR\rs.xml"
Exit Code	0
Error Description	""
GUID	"432640786324026346432"

Table 8. (continued)

Field Name	Value
Statistics	[get:571] [add:571] [err:0]

The Tombstone Manager

The Tombstone Manager will monitor the number of tombstone records at runtime and delete old records as per the values of the `com.ibm.di.tm.autodel.age`, `com.ibm.di.tm.autodel.records.trigger.on`, `com.ibm.di.tm.autodel.records.max` configuration properties (see "Tombstone Manager Configuration").

- The Tombstone Manager tracks Config Instances, AssemblyLines and EventHandlers stop events.
- The Tombstone Manager uses the local Server API calls for registering for event notifications and receiving stop events for Config Instances, AssemblyLines and EventHandlers.
- The Tombstone Manager uses the TDI System Store for data persistence.
- The Server API (documented in the JavaDocs) interfaces contain calls for querying the Tombstone Manager for various data, like AssemblyLine, EventHandler and Config Instance Tombstones.
- The Tombstone Manager provides options for deleting old tombstone records.

A possible AssemblyLine tombstone lifecycle could look like:

- The Tombstone Manager receives a Server API event that an AssemblyLine has terminated (this assumes the Server API and the Tombstone Manager are turned on and the configuration file specifies that tombstones are created for this AssemblyLine)
- The Tombstone Manager extracts the required data from the Server API event and creates a corresponding database tombstone record in the System Store.
- While the tombstone record is available in the System Store queries can be execute through the Server API calls that provide all the information contained in the tombstone record.
- The tombstone record will be deleted from the System Store either when an explicit cleanup Server API call is executed that deletes it or the logic for automatic deletion of old tombstone records collects it. Neither of these is required to happen so theoretically the tombstone record may live forever.

Tombstone Manager Configuration

The Tombstone Manager task is configured by means of properties in the `global.properties` or `solution.properties` file for your Config instance.

Note: In order for the Tombstone Manager to function, the Server API must be switched on; that is, the property `api.on` must be set to **true**.

The relevant properties are:

com.ibm.di.tm.on

Master switch for the Tombstone Manager. Values are **on** and **off** - if set to off, no Tombstones are generated even if specified in the Config file; neither are they managed (nor can they be queried using the Server API, or AMC).

The default value for this property is **false**.

com.ibm.di.tm.autodel.age

The number of days a Tombstone will live. When this property is present and contains an integer value greater than 0 the Tombstone Manager will automatically delete all tombstone records that are older than the specified number of days.

The logic for tombstone record deletion is triggered on TDI Server startup and once a day on a long running TDI Server.

The default value for this property is **0**.

com.ibm.di.tm.autodel.records.trigger.on

Specifies the total number of tombstone records that will trigger the logic for trimming the number of tombstone records to a certain number.

The default value for this property is **10000**.

com.ibm.di.tm.autodel.records.max

The number of Tombstones to be retained once the trigger specified by the previous parameter, `com.ibm.di.tm.autodel.records.trigger.on` is exceeded.

The default value for this property is **5000**.

com.ibm.di.tm.create.all

This property acts as an override switch for the values specified in the Config files. When this property is set to **true**, Tombstone Manager will create Tombstones for every AssemblyLine, EventHandler and Config Instance regardless of the values specified in the configurations. This is useful to turn on Tombstone creation for pre-6.1 configurations that do not have tombstone values without modifying the configurations.

The automatic cleanup logic determined by the `com.ibm.di.tm.autodel.age` property is independent of the automatic cleanup logic determined by the `com.ibm.di.tm.autodel.records.trigger.on` and `com.ibm.di.tm.autodel.records.max` properties.

The Tombstone Manager uses the TDI logging framework and logs its messages in the TDI Server main log.

An example section in the `global.properties` or `solution.properties` file could look like:

```
com.ibm.di.tm.on=true
com.ibm.di.tm.autodel.age=90
com.ibm.di.tm.autodel.records.trigger.on=50000
com.ibm.di.tm.autodel.records.max=25000
com.ibm.di.tm.create.all=false
```

This set of configuration properties specifies that: The Tombstone Manager is turned on. Tombstones older than 90 days will be automatically deleted. Also when the total number of tombstone records reaches 50000, the oldest 25000 tombstone records will be automatically deleted.

Chapter 13. Multiple TDI services

IBM Tivoli Directory Integrator as Windows Service

Introduction

In IBM Tivoli Directory Integrator 6.1.1 there is a mechanism that allows multiple TDI server instances to be registered as Windows services. Each instance requires a separate solution directory. After creating a solution directory, a utility program should be copied in it. The name of the program is "ibmdiservice.exe". The configuration of the utility program and the Windows service is made with a properties file named "ibmdiservice.props". Each solution directory should contain a configuration properties file.

The IBM Tivoli Directory Integrator can be registered and run as a Windows Service on the following Windows Platforms:

- Windows 2000 Server
- Windows 2000 Advanced Server
- Windows XP Pro
- Windows 2003 Std. Ed.
- Windows 2003 Enterprise Ed.

Each Windows service must have a different name. A property called "servicename" in the property file specifies a name that is used in creation of the Windows service name and the Windows service display name. The Windows service name is formed by prefixing the value of the "**servicename**" property with the "ibmdisrv-" prefix. The Windows service display name is formed by inserting the value of the "servicename" property between the brackets of "IBM Tivoli Directory Integrator ()". For example if the "servicename" property is set to "test" the Windows service name will be "ibmdisrv-test" and the Windows service display name will be "IBM Tivoli Directory Integrator (test)". If the "servicename" property is not present or has no value default names are used. The default names for the Windows service name and the Windows service display name are "ibmdisrv" and "IBM Tivoli Directory Integrator".

A property exists so it can be configured whether the Windows service is started automatically on Windows startup or has to be started manually. The name of the property is "**autostart**" and the valid values for it are "true" and "false".

Note: This property is used during installation and uninstallation as well as while the service is running. That is why the property value must not be changed after the Windows service has been installed.

For more information about the TDI Windows service configuration properties file see the ""Configuring the service" on page 163" section.

Installing and uninstalling the service

Installing the service

Do the following to install the IBM Tivoli Directory Integrator service:

1. Make sure the IBM Tivoli Directory Integrator is installed. The install folder of the IBM Tivoli Directory Integrator is referred to as *root_directory*. See 9.
2. Choose a solution folder that will be used by IBM Tivoli Directory Integrator when it is started as a Windows service - this can be any folder of your choice. Once IBM Tivoli Directory Integrator is installed as a service the solution folder used by the service cannot be changed until it is uninstalled

as a service. Note that choosing the solution folder for the Windows service does not prevent from running IBM Tivoli Directory Integrator from the with any other solution folder.

3. Once the solution folder is chosen copy into that folder all files from the `root_directory/win32_service` folder: these are ″ibmdiservice.exe″, ″ibmdiservice.props″ and ″log4j.properties″.
4. Execute the following command from the solution folder chosen for the Windows Service:

   ```
   ibmdiservice.exe -i
   ```

Uninstalling the service

Note: In order to use the TDI 6.1.1 version of the “ibmdiservice.exe” utility program any registered pre-TDI 6.1.1 Windows service must be uninstalled and then the TDI 6.1.1 windows service must be installed. This is necessary because the TDI 6.1.1 windows service uses a different default name for the Windows service name – “ibmdisrv” as opposed to the pre-TDI 6.1.1 default name of “IBM Tivoli Directory Integrator”.

Do the following to uninstall the IBM Tivoli Directory Integrator service:

1. Make sure the IBM Tivoli Directory Integrator service is stopped.
2. Execute the following command from the solution folder chosen when installing the service:

   ```
   ibmdiservice.exe -u
   ```

Notes:

1. Uninstalling the IBM Tivoli Directory Integrator service does not uninstall the IBM Tivoli Directory Integrator itself. You can still use the IBM Tivoli Directory Integrator but it is not registered and run as a Windows service. You can install IBM Tivoli Directory Integrator service again later.
2. If the IBM Tivoli Directory Integrator service is installed and you wish to completely uninstall the IBM Tivoli Directory Integrator (not just the service), do the following:
   a. Uninstall the Windows service.
   b. Uninstall the IBM Tivoli Directory Integrator (Refer to Uninstalling on Windows).

Starting and stopping the service

The IBM Tivoli Directory Integrator service automatically starts the IBM Tivoli Directory Integrator at system boot. The IBM Tivoli Directory Integrator is not, however, automatically started when the service is installed. After installing the service you have two options to start the service:

- Reboot the machine.
- Start the IBM Tivoli Directory Integrator service from the Windows Services panel.

Manual start and stop

You can manually start and stop the IBM Tivoli Directory Integrator service from the Windows Services panel.

In the **Services** panel you must select the service IBM Tivoli Directory Integrator and, depending on the Windows version, either click the **Start/Stop** button, or right-click on the service name and select **Start/Stop**.

Changing service startup type

By default, the IBM Tivoli Directory Integrator service is configured to start automatically on system boot.

You can manually change the service startup mode from the Windows Services panel to **Manual** or **Disabled**.

Logging

The IBM Tivoli Directory Integrator service logs all messages (**error**, **info** and **debug**) in the Application Windows system log. You can view these messages with the Windows Event Viewer.

Configuring the service

The IBM Tivoli Directory Integrator service is configured through the ibmdiservice.props file placed in the solution folder chosen during the installing the service.

Note: Before running the service, make sure this file is properly configured as described in this section. The service might fail if the file contains incorrect values.

The following properties are specified in the ibmdiservice.props file:

path Specifies the PATH environment variable used for running the IBM Tivoli Directory Integrator process (this property is usually the same as the PATH variable from ibmdisrv.bat, but you can change it). This is an optional property.

ibmdiroot
Specifies the root folder of the IBM Tivoli Directory Integrator (for example, `C:\Program Files\IBM\IBMDirectoryIntegrator`). This is a required property.

configfile
Specifies the file path to the IBM Tivoli Directory Integrator configuration file. This is an optional property.

assemblylines
Specifies in a comma-delimited format the AssemblyLines that are started automatically when the IBM Tivoli Directory Integrator service is started. This is an optional property.

eventhandlers
Specifies in a comma-delimited format the EventHandlers that are started automatically when the IBM Tivoli Directory Integrator service is started. This is an optional property.

cmdoptions
Specifies other options that are directly passed to the IBM Tivoli Directory Integrator on service startup (see "IBM Tivoli Directory Integrator options" in *IBM Tivoli Directory Integrator 6.1.1: Users Guide* for the full list of IBM Tivoli Directory Integrator options).

One such option could be the **-c** option; here you could specify multiple config files (separated by commas), something which is not allowed by the **configfile** parameter.This is an optional property.

debug Specifies **true** or **false** to correspondingly turn debug information on or off. When debug information is turned on, detailed trace messages are dumped in the Application Windows system log. This is an optional property.

Note: When specifying properties in the configuration file, specify each property on a single line and use the following format:

```
<property_name>=<property_value>
```

There must be no spaces around the equals (=) sign.

An example of a completed `ibmdiservice.props` file looks like the following:

```
path=C:\Program Files\IBM\TDI\V6.1.1\jvm\jre\bin;
C:\Program Files\IBM\TDI\V6.1.1\libs;
ibmdiroot=C:\Program Files\IBM\TDI\V6.1.1
configfile=rs.xml
assemblylines=AssemblyLine1,AssemblyLine2
eventhandlers=EventHandler1,EventHandler2
cmdoptions=-d
debug=false
```

Note: If you change any of the properties in `ibmdiservice.props`, you must restart the service for the changes to take effect.

IBM Tivoli Directory Integrator as Linux/UNIX Service

Deployment methods

On Linux and UNIX platforms, there are two different ways of ensuring that certain system jobs or 'daemons' start and stop at respectively system initiation and system termination:

1. Using a script in */etc/init.d* containing the logic to start and stop the daemons you are interested in. This script you then (hard)link to scripts in */etc/rc3.d*: their names beginning with `SXX...` and `KXX...` - the XX being a numeral which causing the files to show up in the right sequence in the /etc/rc3.d directory. The scripts starting with S are called when the system reaches run phase 3 at system startup, and the scripts starting with K are called when the system terminates.
2. By editing the */etc/inittab* file.

The latter process is what we will describe here. Some of the information presented could be used to construct scripts utilizing the first deployment method.

Tailoring /etc/inittab

In order to start up TDI daemon processes when the UNIX/Linux OS starts appropriate entries must be added to the */etc/inittab* file. The registering of TDI as a windows service on Windows translates to adding a line of text to the */etc/inittab* file on UNIX/Linux. The un-installation of the TDI windows service on Windows translates to removing the corresponding entries from the */etc/inittab* file. For each TDI daemon process that needs to be started on system startup one line of text must be added to the */etc/inittab* file. The format and meaning of the entries in this file is described below. Each entry in the */etc/inittab* file has the following format:

Identifier:RunLevel:Action:Command

A description of each of these fields is as follows:

- The **Identifier** field is a string (at least a single character in length) that uniquely identifies an object. This string is used to uniquely identify the corresponding .
- The **RunLevel** field is the run-level in which this entry can be processed. Run-levels effectively correspond to a configuration of processes in the system. Each process started by the init command is assigned one or more run-levels in which it can exist. A run-level is represented by the numbers 0 through N, where N is a positive integer different for the different UNIX/Linux operating Systems (for example on some AIX machines N is 9, on RedHat Linux N is 6, etc.). If the OS is running in run-level 3, for example, then only processes specified for run-level 3 are started.

  The **RunLevel** field can define multiple run-levels for a process by selecting more than one run-level in any combination from 0 through N. For example, if TDI needs to run in run-level 3 and 6, then the run-level must be specified as “36”. If no run-level is specified, the process is assumed to be valid at all run-levels.

  It is recommended that no run-level numbers are specified, unless the specific TDI solution specifically needs to.
- The **Action** field is a value from a set of predefined actions which tells the **init** command how to treat the process specified in the **Command** field. There are many actions recognized by the init command, but for running the TDI server as a daemon process it is recommended that the **once** action be used. The semantics of the **once** action are:

  When the init command enters a run-level that matches the entry’s run level, start the process, and do not wait for its termination. When it dies, do not restart the process. When the system enters a new run level, and the process is still running from a previous run level change, the program will not be restarted. All subsequent reads of the /etc/inittab file while the init command is in the same run level will cause the init command to ignore this entry.
- The Command field specifies the shell command to execute.

Here are three example TDI-related entries in /etc/inittab:

```
tdi1::once:/opt/IBM/TDI611_1/ibmdisrv –c "/opt/IBM/TDI611_1/myconfigs/rs1.xml" –r "testAL1"
tdi2::once:/opt/IBM/TDI611_2/ibmdisrv –c "/opt/IBM/TDI611_2/myconfigs/rs2.xml" –r "testAL2"
tdi3::once:/opt/IBM/TDI611_3/ibmdisrv –c "/opt/IBM/TDI611_3/myconfigs/rs3.xml" –r "testAL3"
```

This example starts three TDI server instances which are installed in different folders.

Note: There are some differences in the different UNIX/Linux operating systems in regard to system startup. That is why the information provided here covers the main issues of starting TDI on a UNIX/Linux system and does not refer to any specific UNIX/Linux system.

As an example of an /etc/inittab file, detailed information about the /etc/inittab configuration file for an AIX system can be found at http://publib16.boulder.ibm.com/pseries/en_US/files/aixfiles/inittab.htm

IBM Tivoli Directory Integrator as i5/OS Service

TDI can be started as a service on i5/OS (OS400) by defining it as a custom TCP/IP server. The server can then either be auto started when TCP/IP is started or manually started via the **STRTCPSVR** command. To define a TCP/IP server, the **ADDTCPSVR** (Add TCP Server) command is used which requires a program to be registered that will be called by the **STR/ENDTCPSVR** command.

The following are the steps to manually start a TDI service which will start an AssemblyLine "AL" in Config file `rs.xml` .

1. Create Library for TDI:

   ```
   CRTLIB (TDILIB) TYPE(*TEST) TEXT('TDI Library')
   ```

2. Create a Source physical file in the TDILIB library:

   ```
   CRTSRCPF FILE(TDILIB/TDISRC)
   ```

3. Add a file member to the Source file and add the following code using the Source Entry Utility (SEU):

   ```
   PGM PARM(&P)   /* START TDI Service Program */
   DCL VAR(&P) TYPE(*CHAR) LEN(10)
   SBMJOB CMD(STRQSH  CMD(' <TDI_ INSTALLATION >/ibmdisrv -c +
     <TDI_INSTALLATION> /configs/rs.xml -r  AL ')) +
                              JOB(TDIService) ALWMLTTHD(*YES)
   ENDPGM
   ```

   The program is saved as **TDIPGM** in **TDILIB**.

4. Create a program object from the file member in the TDILIB library. (Use **CRTBNDCL** cmd or Option 3 in Programmer menu)

5. Add the TDI TCP/IP server using the **ADDTCPSVR** command with name *TDISERVER*:

   ```
   ADDTCPSVR   SVRSPCVAL(*TDISERVER)
               PGM(TDILIB/TDIPGM)
               SVRNAME(TDISERVER)
               SVRTYPE(TDISERVER)  AUTOSTART(*NO)
   ```

6. Start the TDI TCP/IP server:

   ```
   STRTCPSVR   SERVER(*TDISERVER)
   ```

For more information on the i5/OS (OS400) (V5R4) and commands that have been demonstrated above, refer to http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=/clfinder/finder.htm

Chapter 14. z/OS environment Support

A subset of IBM Tivoli Directory Integrator is available in the native z/OS environment (version v1.6, v1.7 and v1.8), with IBM JVM 1.5. You can start an instance of the Server by executing the startup shell script `usr/lpp/idti/ibmdisrv` residing under the Unix System Services.

The user whose identity the TDI Server will run under needs an OMVS segment definition in his profile specifying that at least 200MB operating memory is available.

The available TDI components are:

- AssemblyLine Connector
- ACT Connector
- Command Line Connector
- File System Connector
- LDAP Connector
- LDAP Server Connector
- IDS Changelog Connector
- System Store Connector
- Timer Server Connector
- HTTP Client Connector
- HTTP Server Connector
- JDBC Connector
- JMS Connector *
- JMX Connector
- JNDI Connector
- RAC/GLA Connector
- GLA Connector
- RDBMS Changelog Connector
- Script Connector
- Server Notifications Connector
- SNMP Connector
- SNMP Server Connector
- System Queue Connector *
- TCP Connector
- TCP Server Connector
- TIM DSMLv2 Connector
- z/OS Changelog Connector
- Web Service Receiver Server Connector
- Axis Easy Web Service Server Connector
- Castor XML to Java Function Component
- Castor Java to XML Function Component
- DSML Parser
- DSMLv2 Parser
- Simple Parser

- SPMLv2 Parser
- CVS Parser
- XML Parser
- XML SAX Parser
- HTTP Parser
- LDIF Parser
- DSMLv2 EventHandler
- IDS EventHandler
- HTTP EventHandler
- AssemblyLine Function Component
- Axis Java To Soap Function Component
- CBE Generator Function Component
- WrapSoap Function Component
- InvokeSoap WS Function Component
- Axis Soap To Java Function Component
- Axis EasyInvoke Soap WS Function Component
- Complex Types Generator Function Component
- EMF XMLToSDO Function Component
- EMF SDOToXML Function Component
- Java Class Function Component
- SendEMail Function Component
- z/OS TSO Command Line Function Component
- Remote Command Line Function Component

* When TDI runs on z/OS the JMS Connector and the System Queue cannot use WebSphere MQ as the JMS provider because of a limitation in WebSphere MQ (WebSphere MQ doesn't support client code on z/OS).

In addition to the components listed above, the System Store (using Cloudscape, or configured to use DB2) is supported on z/OS.

The z/OS TSO Command Line Function Component is of particular relevance for the z/OS environment. It is able to execute privileged z/OS TSO Commands. This component addresses the need to manage RACF®, TopSecret and ACF2 users – this can be achieved by executing TSO commands.

The Config Editor and AMC, however, are not supported natively on z/OS; instead, you should use remote management options, like

- The Remote Config Editor. Run the Config Editor on a supported platform, and access Config files on z/OS using a configured Config Instance on z/OS.
- The Administration and Monitor Console
- Applications that use the remote TDI Server API.

Using the Remote Config Editor on z/OS

On z/OS the only way of editing and modifying Config files stored on z/OS is by using the Remote Config Editor. See "Using the Remote Config Editor" on page 79 for some general characteristics for this particular way of editing Config files. In addition to that, there are some additional considerations when using the Remote CE for z/OS:

1. Config files developed locally need to be uploaded (using FTP) in binary mode to the z/OS machine.

2. The default encoding on z/OS is EBCDIC (or IBM-1047 as it is also known). It is very different from ASCII or from UTF-8. There are no common character ranges between it and ASCII/UTF-8 (whereas the range 0-127 is the same between ASCII and UTF-8).

   This is why any text in EBCDIC viewed as ASCII/UTF-8 looks unintelligible and vice versa.

   Any file created on a Windows or Unix machine (for example, a properties file, a text file, etc.) that needs to be read by TDI running on a z/OS box must follow the native encoding format of z/OS. One way of converting a Windows file to the native z/OS encoding format is to use the encconvz utility shipped with TDI in the ″tools″ directory.

   Here is an example (UNIX) usage of the encconvz command:

   ```
   ./encconvz myfile_win.txt   myfile_z/OS.txt    ISO-8859-1   IBM-1047
   ```

   For more information, see "Handling configuration and properties files."
3. CLI (tdisrvctl) The tdisrvctl remote utility is installed in the <TDI_root_directory>/bin directory.

   Note that for z/OS, by default, the tdisrvctl utility is configured to create logs in the `<TDI_root_directory>/logs/tdisrvctl.log` file. Since this could be a read-only location, it is recommended that you edit the `<TDI_root_directory>/etc/tdisrvctl-log4j.properties` file to point its log file to a writable location. The property to edit in the tdisrvctl-log-4j.properties file is: `log4j.appender.Default.file`. Also, if the **-h** (hostname) option is skipped, the tdisrvctl utility takes *localhost* as the default. This may not work on z/OS for all cases. Always specify the -h option with the machine's IP address.

Handling configuration and properties files

Handling of configuration and properties files is important because of the specific default encoding used on z/OS (EBCDIC), which is not compatible with UTF-8 usually used on other platforms.

The TDI Server can read configuration files in any encoding that is supported by the JVM; TDI Configuration files are read with the encoding specified in the header of the XML file. If no encoding is specified in the header of the configuration file, **UTF-8** is used.

The TDI Server can write configuration files in any encoding that is supported by the JVM.

- If the **-n <encoding>** switch is used when starting the TDI Server the encoding specified by <encoding> will be used for writing configuration files.
- If the **-n** switch is not specified and the system property `com.ibm.di.config.encoding` is non-null then the value of this property is used as encoding when writing configuration files.
- If neither of the **-n** switch nor the `com.ibm.di.config.encoding` system properties are specified, then **UTF-8** is used for writing configuration files.

In all cases the encoding used for saving the XML configuration file is written in the header of the XML file.

This strategy for reading and writing configurations assumes that it is usually the UTF-8 encoding that will be used on z/OS for configuration files. If however you want to use a different encoding (for example the system default so that the configuration file can be opened by a text editor like vi) then you are provided with a mechanism that can be used to create and use configuration files with an arbitrary encoding.

You should pay attention on the encoding used whenever you operate with text files on the z/OS system. For example when a file is read with the FileSystem Connector the **Character Encoding** parameter of the Parser used should specify the encoding of the file or should be left empty when the file uses the default EBCDIC encoding.

All *.properties files (e.g. global.properties, log4j.properties, etc.) in the installation directory and/or Solution Directory are read with the system default encoding. This makes it convenient for you to open and edit the properties files directly on the z/OS system.

ASCII mode

The `ibmdisrv` startup script starts the TDI server without altering its default encoding, which on z/OS is EBCDIC (IBM1047). In order to run the TDI server on z/OS in ASCII mode you need to start it using the `ibmdisrv_ascii` startup script. This script starts the TDI server with its default encoding set to ASCII (ISO-8859-1)

Note: In ASCII mode, the TDI server ignores the global.properties file. Only the `solution.properties` file in your Solutions Directory is used, and this file needs to be encoded in the ASCII character set. For an overview of files in the Solution Directory, see "Solution Directory files" on page 30.

Encoding of solution.properties
Altering the default encoding of the TDI server affects how the `solution.properties` file is read. That is why the encoding of the `solution.properties` file in your Solution Directory must be changed to ASCII before starting the TDI server in ASCII mode. The location of the solution.properties file is `"<YOUR_SOLUTION_DIRECTORY>solution.properties"`.

Changing the encoding of a text file on z/OS
The standard `iconv` utility available on z/OS can be used to convert the encoding of a text file. Starting the iconv utility with no parameters on the z/OS prints usage information.

The global.properties file
When the TDI Server on z/OS is run in ASCII mode the `global.properties` file is ignored and only the `solution.properties` file in your Solution Directory is read. That is why you need to have all the required properties for your solution in the `solution.properties` file in your Solution Directory.

Log files
When the TDI Server on z/OS is run in ASCII mode the server log files are encoded in ASCII when being written to the file system. That is why in order to read these ASCII log files you might need to first convert their encoding to the native encoding on z/OS, which is EBCDIC (IBM1047).

Console output
When the TDI Server on z/OS is run in ASCII mode any text output to the z/OS console by the server appears garbled. This is caused by the output text being encoded in ASCII while the console expects the text to be encoded in EBCDIC. In order to read the server output to the console, the server output can be redirected from the console to a file and then this file can be converted from ASCII encoding to native encoding on z/OS (EBCDIC).

Appendix A. Dictionary of terms

IBM Tivoli Directory Integrator terms

Action Manager (AM)
Action Manager is a stand-alone Java application used to configure failure-response behavior for TDI 6.1.1 solutions. AM executes *rules* defined with AMC v.3. An AM rule consists of one or more *triggers* that define a "failure" situation – such as the termination of an AL that should not stop running, or if an AL has not been executed within a given time period, and so forth Furthermore, each rule also defines *actions* to be carried out in case of this "failure". Actions include operations like sending events or e-mail, starting ALs (locally or remotely) and changing configuration settings. Action Manager requires TDI 6.1.1 and AMC v.3.

Accumulator
A special object that can be set in a Task Call Block (TCB) for use when starting another AssemblyLine either via a scripted call, or a component like the AssemblyLine Connector or the AssemblyLine FC. The Accumulator is either a collection of Work Entry objects handled by the called AL, or it is a component that is called to output each Entry. Accumulator handling is done at the end of each AssemblyLine Cycle.

Adapter
Adapter is a word is used in many contexts and with different meanings. A *TDI Adapter* refers to an AssemblyLine that is "packaged" as a single Connector. Creating a TDI Adapter requires setting up an AssemblyLine that is written to perform (and expose) one or more business related tasks. Each task is defined as an AssemblyLine Operation (for example, ‘EnableAccount’, or ‘ReturnGroupMembers’). This AL can then be *published* for sharing, and can be leveraged by the AssemblyLine Connector which offers mode settings reflecting these operations[2].

AL Shorthand for AssemblyLine.

Administration and Monitoring Console (AMC)
AMC is a browser-based console for managing and monitoring TDI solutions. AMC version 3, which is part of the *TDI 6.1* and later releases, runs on the WebSphere Application Server (enterprise and express versions), as well as Tomcat. Each AMC version is designed to work with a specific release of TDI and is incompatible with other versions. AMC v.3 is designed for *TDI 6.1*, AMC v.2 works with *TDI 6.0* and AMC v.1 runs with *TDI 5.2*.

API Application Program Interface. A way of programmatically (local or networked) calling another application, as opposed to using a command-line or a shell script.

Appender
Appender is a log4j term (a third party Java library) for a module that directs log-messages to a certain device or repository. In IBM Tivoli Directory Integrator you control logging for your AssemblyLines by creating and configuring *Appenders*, either under the Logging tab of a specific AL, or under Config -> Logging in the Config Browser to control how all AssemblyLines in the Config do their logging.

AssemblyLine (AL)
The basic *unit-of-work* in a TDI solution. Each AL runs as a JVM thread in the Server and is made up of a series of AssemblyLine components (one or more Connectors, Functions, Scripts, Attribute Maps and Branches) linked together and driven by the built-in workflow of the AssemblyLine.

[2] AL Operations are also accessible via the AssemblyLine FC.

AssemblyLine Component

This term denotes an TDI component used to construct AssemblyLines. The possible Components are:

- Connectors
- Function Components
- Script Component
- Attribute Map Component
- Branches (including Loops and Switches)

The components list in an AssemblyLine is divided into two sections: *Feeds* where the Work Entry for each AL cycle is created from input data by a Connector in Iterator or Server mode, and the *Flow* section that holds the Connectors (in any mode except Server), Functions, Attribute Maps and Scripts providing the additional data access and processing.

AssemblyLine Operation

A business task that is implemented by an AssemblyLine and published via its Operations tab. Each Operation can have its own Input and Output Attributes Maps for defining the parameters expected when this Operation is invoked (Input Map), as well as those returned (Output Map). This is also called the *Schema* of the Operation.

AssemblyLine Phases

An AssemblyLine goes through three phases:

Initialization

At this point the TDI Server uses the "blueprint" for the AssemblyLine in the Config to create the various components as well as set up the AL environment, including processing the TCB, starting the ALs script engine and invoking the AssemblyLine's Prolog Hooks. All components that are configured for Initialization At Startup are initialized at this point causing their Prolog Hooks to get run as well.

Cycling

Now the AL workflow drives each of its components in turn, starting each cycle by invoking the On Start of Cycle Hook. Then the currently active *Feeds* Connector reads in data, creates the Work Entry and passes it to the *Flow* section. The Work Entry is passed from component to component until the end of *Flow* is reached, at which time control is returned to the start of the AssemblyLine again[3]. Cycling continues until an unhandled error occurs or there is no more data available (for example, the Iterator reaches End-of-Data).

Shutdown

When cycling stops then the AssemblyLine goes into Shutdown phase: Epilog Hooks are called and all initialized components are closed down (which flushes output buffers and executes their Epilog Hooks as well). Finally the AssemblyLine closes down its environment and its thread terminates.

AssemblyLine Pool

Actually a collection of AL *Flow* sections that can be configured to allow a Server mode Connector to service more clients. Available for ALs that use Server mode Connectors and set up in the AssemblyLine's Config tab.

Attribute

Part of the TDI Entry data model. Attributes are carried by Entry objects (Java "buckets", like the Work Entry) and they can hold zero or more *values*. These *values* are the actual data values read from, or written to connected systems, and are represented in TDI as Java objects.

[3] If the current cycle was fed by a Server mode Connector, then the reply is created by the Server mode Connector's Output Map and sent to the client.

Attribute Map (AttMap)

An Attribute Map is a list of rules (individual Attribute Mapping instructions) for creating or modifying Attributes in an Entry object typically based on the values of Attributes found in another Entry object. Components like Functions and Connectors have an Input Map for taking data read into local cache (the conn Entry) and use this to define Attributes in the Work Entry. These components also have an Output Map that takes Attributes carried by the AssemblyLine (in its Work Entry) and use this to set up the conn Entry that will be used by the component's output operation. Attribute Map components use the Work Entry as both the source and target of the mappings.

Attributes can be mapped in one of three ways: Simply (copying values between Attributes), Advanced (using a snippet of JavaScript) or with a TDI Expression.

Attribute Map component

A free-standing list of individual Attribute Mappings that take values from the Work Entry and use them to create and update other Attributes in the Work Entry. They can be tied to Connector and Functions to define their Input or Output Maps. Note that Input and Output Maps can be copied to the library as AttMap components for reuse.

Best Practices

Recommended methodology and techniques for working with TDI. These include the ABCs: Automation, Brevity and Clarity:

Automation

Use the automated features of TDI in preference to your own custom scripted logic whenever possible – for example, using Branches/Loops instead of extensive scripting in Hooks. Not only will this make your solution easier to read and maintain (and step through with the AL Debugger!), but your solution will benefit directly as built-in logic is strengthened and extended with each new release.

Brevity

Keep your AssemblyLines as short and simple as possible, as well as your script snippets. Break complex logic into simpler patterns that can be tested individually and reused in other solutions.

Clarity

Choose legibility over elegance. Write solutions for others to read and maintain.

Branches

A construct used to control the flow of logic in an AssemblyLine. TDI 6.1.1 provides three types of Branches:

- Simple Branches (IF, ELSE-IF and ELSE)
- Loops (Connector-based, Attribute-based or Conditional)
- Switches (for example, switching on the Work Entry delta operation code, or the Operation an AL is called with).

CBE Common Base Event. A term used in the Common Base Infrastructure. See "Common Base Event" in the chapter about the CBE Generator Function Component in the *IBM Tivoli Directory Integrator 6.1.1: Reference Guide*.

CEI The IBM Common Event Infrastructure. See "The Common Event Infrastructure", in *IBM Tivoli Directory Integrator 6.1.1: Reference Guide*.

Change Detection Connector (CDC)

A Connector that returns changes made in the connected system. Typically, a CDC can be configured to return only a subset of Entries: new, modified, deleted, unchanged or a combination of these. Some CDC's provide only the changed Attributes in the case of a modified

Entry, while others return them all. Change Detection Connectors also tag the data with special *delta operation codes* to indicate what has changed, and how.[4].

CloudScape (Derby)
A free, Java-based Relational Database, akin to IBM DB2, that is bundled with TDI as the default repository for the System Store.

CLI Command-line Interface, such as the `tdisrvctl` utility.

Components
The architecture of IBM Tivoli Directory Integrator is divided into two parts: generic functionality and technology-specific features. Generic functionality is provided by the TDI *kernel* which provides automated behaviors to simplify building integration solutions. The kernel also lets you extend or override these behaviors as desired, as well as doing the housekeeping for your solution: logging/tracing, Hooks for error handling, API and CLI access, and so forth Technology-specific "intelligence" is handled by helper objects called *components*, such as Connectors, Functions, Branches, Scripts and Attribute Map components. Components provide a consistent and predictable way to access heterogeneous systems and platforms, and the kernel lets you "click" together components to build AssemblyLines.

Compute Changes
A special feature of the Connector Update mode that instructs the Connector to compare the Attributes about to be written to the connected system with those that exist in this data source already – in other words, it compare the value of each Attribute in the conn Entry (the result of the Output Map) with the corresponding ones found during the Update mode *lookup* operation (which is stored in the current Entry).

Config or Config File
A collection of AssemblyLines and components that comprise a solution. A Config is stored in XML format, typically in a Config file and is written, tested and maintained using the Config Editor.

Config Browser
This is the tree-view window at the top left-hand part of the Config Editor screen. It gives you access to Config-wide settings, the AssemblyLines and components that make up the Config, as well as Properties, *included* Configs and custom Java libraries that are to be loaded and made available to your scripts.

Config Editor (CE)
The graphical development environment used to write, test and maintain Configs. Configs are stored in XML format and are deployed by assigning them to one or more IBM Tivoli Directory Integrator Servers to execute.

Config Instance
A copy of a TDI Config that is running on a Server. Typically loaded only once on a given Server, TDI allows you to start the same Config multiple times if desired. Each running copy is given its own context and can be accessed individually through the API.

Config View
This term is used in the context of AMC to describe how a particular Config appears in the management screens of AMC. A Config View is a selection of the AssemblyLines and properties that are to be visible onscreen (user/role based), providing solution-oriented Config administration and management. Config Views can be combined to define a Monitoring View in AMC.

conn Entry
This is the local Entry object maintained by a Connector or Function. The conn Entry is used as a

[4] For LDAP there is also a special kind of modify operation where the directory entry has beeen moved in the tree: *modrdn*, i.e. a "renamed" entry.

local cache for read and write operations, and data is moved between this cache and the AssemblyLine's Work Entry via Attribute Maps (specifically, Input and Output Maps).

Connector
One of the component types available in TDI to build AssemblyLines. Connectors are used to abstract away the technical details of a specific data store, API, protocol or transport, providing a common methodology for accessing diverse technologies and platforms.

Unlike the other components, Connectors can perform different tasks based on their *mode* setting (for example, Iterate, Delete, Server and Lookup). Modes are provided by the AssemblyLine component part of the Connector. However, the list of modes supported is dependent on the Connector Interface.

Connector Interface
When a component is used in an AssemblyLine, a distinction must be made between the *Connector Interface* (CI), containing the "intelligence" for working with a connected system (for example, LDAP, JDBC, Notes, and so forth), and the *AssemblyLine Connector*. [5]. This latter object is the "AL wrapper" that allows the CI to be plugged into an AssemblyLine and provides them with a consistent set of generic features, like input or output maps, Link Criteria, Hooks and the Delta Engine. See ″Objects″ in *IBM Tivoli Directory Integrator 6.1.1: Reference Guide* for more information. See also ″Connectors″ in *IBM Tivoli Directory Integrator 6.1.1: Reference Guide*.

Connector Pool
Unlike the AssemblyLine Pool feature available to ALs using Server mode Connectors, a Connector Pool is a global collection of pre-initialized Connectors that can be used in multiple ALs. Note that the Connector Initialization setting "Initialize and terminate every time it is used" means that no AssemblyLine gains exclusive rights to a pooled Connector, giving you detailed control over resources used by your solution.

current Entry
This Entry object is local to a Connector Interface (just like the conn Entry) and contains the Attributes read in from a *lookup* operation (for example, as carried out by Lookup, Update and Delete modes). It is used to provide the Compute Changes feature.

Delta Engine
Available for Connectors in Iterator mode, the Delta Engine provides functionality for detecting changes in data sources that do not offer any changelog or change notification features. See Delta Operation Codes, as well as ″Deltas and compute changes″ in *IBM Tivoli Directory Integrator 6.1.1: Users Guide* for more information.

Delta mode (for Connectors)
This Connector mode is used to the apply changes specified with delta operation codes in the Work Entry, and to do so as efficiently as possible by performing incremental modifications. Note that Delta mode is only available for the LDAP and JDBC Connectors, and will not work with Entries without a valid delta operation code. See ″Deltas and compute changes″ in *IBM Tivoli Directory Integrator 6.1.1: Users Guide*.

Delta Operation Codes
These are special values assigned to Entries, Attributes and their values to reflect change information detected in some data source. An Entry that has delta codes assigned is called a *Delta Entry*, and these are only returned by a limited set of components: Change Detection Connectors, the Delta Engine and the DSML and LDIF Parsers[6]. Delta Operation Codes can be queried and used in Branch Conditions or your own JavaScript code, and are used by Delta mode to apply all types of changes to target systems as efficiently as possible.

See also ″Deltas and compute changes″ in *IBM Tivoli Directory Integrator 6.1.1: Users Guide*.

[5] Functions are similar to Connectors in that they are divided into two parts: the Function Interface and the AssemblyLine Function. Unlike Connectors, Functions have no mode setting.

[6] Note that these Parsers only return Delta Entries if the DSML or LDIF entries read contain change information.

Derby CloudScape v10, also known as Apache Derby is a small footprint relational database implemented entirely in Java. Cloudscape is shipped as the default system store for TDI.

Distinguished Name (DN)
An LDAP term that refers to the fully qualified name of an object in the directory, representing the *path* from the root to this node in the directory information tree (DIT). It is usually written in a format known as the User Friendly Name (UFN). The dn is a sequence of *relative distinguished names* (RDNs) separated by a single comma (,).

Entry An Entry is a TDI object used to carry data, and forms the core of the TDI Entry model. The Entry object can be thought of as a "Java bucket" that can hold any number of Attributes, which in turn carry the actual data values read from, or written to connected systems. Each Entry corresponds to a single row in a database table/view, a record from a file or an entry in a directory (or similar unit of data), and there are a number of named Entry objects available in the system. The Work Entry and conn Entry are the most commonly used ones, but there is also a current Entry available in some Connector modes, an error Entry that contains the details of the last exception that occurred, and an Operation Entry (Op-Entry) for accessing details of an AL operation.

Epilog A set of Hooks that, if enabled, are run during the AssemblyLine Shutdown phase. Note that the shutdown of components occurs between the two AL Epilog Hooks, which means that the Epilog Hooks of these components are all completed before the AssemblyLine Epilog - After Close Hook is called.

Error Entry
An Entry object that is created by an AssemblyLine during initialization, and contains Attributes like "status", "connectorname" (applies for all types of components) and "exception"[7]. See also Error Handling.

Error Handling
Error Handling in TDI is based on the concept of *exceptions*. Exceptions are a feature of a programming language, like Java, C and C++, that lets you build error handling like a wall around your program. It also lets you fortify smaller parts within any wall, so you can add specific handling where necessary. TDI leverages the power of exception handling so that you can design the error handling in your solution the same way.

First you have the AssemblyLine's On Failure Hook which is called if the AL stops due to an unhandled exception[8]. This is the outer line of defense[9]. The next level is a component, given that it provides Error Hooks. Connectors actually provide two levels of handling: the mode-specific Error Hook, as well as the Default On Error (same goes for Success Hooks as well).

Finally, in your JavaScript code you can do exception handling yourself. Use the `try-catch` statement, for example:

```
try {
      myObj = someFunctionCallThatCanThrowAnException();
} catch ( excptThrown ) {
      task.logmsg("**ERROR - The call failed: " + excptThrown );
}
```

ERP Enterprise Resource Planning, usually indicates a software suite of programs that aims to manage enterprise resources, usually after heavy customization by the software vendor.

[7] The "exception" Attribute holds the actual Java exception object, in the case of an error – in which case the "status" Attribute would also be changed from a value of "ok" to "error" and "message" would contain the error text.

[8] An "unhandled" error is one that is *caught* in an enabled Error Hook (no actual script code is necessary). If you wish to escalate an error to the next level of error handling logic, you need to re-`throw` the exception:

```
throw error.getObject("exception");
```

[9] If you want to share this logic (or that in any Hook) between AssemblyLines, implement it as a function stored inScript and then include them as a Global Prolog for the AL.

EventHandler

EventHandlers are components that reside outside of AssemblyLines, but that were used in older versions of TDI to "listen" for a specific event, and then dispatch this event data to one or more ALs. Each event (like a received DSML message, or a new changelog entry) resulted in a new AssemblyLine being launched, including the setting up and breaking down of all connections—which was quite resource-intensive. The functionality provided by EventHandlers is now handled using Connectors in Iterator or Server mode.

Note: EventHandlers are deprecated as of the TDI 6.1.1 release, although they are still supported for pre-6.1 Configs. Use Connectors in Server and Iterator Modes instead of EventHandlers. From TDI 6.1 on, if you create a new Config, no folder called "Event Handlers" will be visible in the Config Browser window of the Config Editor.

For Configs created prior to TDI 6.1, if you open such a Config, the Config Browser will contain the "EventHandler" library folder, and the TDI Server still supports their operation.

Exception

See Error Handling.

External Properties

A type of Property Store that uses a flat file for storing configuration settings (like passwords and other component parameter settings) outside the Config itself.

Feeds

This is the first section of an AssemblyLine and can only hold Iterator and Server mode Connectors. The Feeds section is where the Work Entry is created from data retrieved from a connected system or client. The Feeds section is like a built-in Loop that drives the Flow section components list, once for each Entry read.

Flow

This is the second (and usually the main) section of an AssemblyLine and holds a list of components; any type, except Connectors in Server mode. The Flow section receives a Work Entry from the currently active Feeds Connector and passes it from component to component for processing.

Function component (FC)

One of the component types available in TDI to build AssemblyLines. Functions are used to abstract away the technical details of a specific service or method call. Typical examples are the AssemblyLine FC used to execute ALs and the Java Class FC that lets you browse jar files and call class methods. Unlike Connectors, FCs do not have mode settings.

Global Prolog

This is a Script component that is defined in the "Scripts" library folder of the Config Browser, and which is configured to be executed when an AssemblyLine starts up. The simplest way to do this is to select which Scripts to use with the "Include Addition Prologs - Select" button. Note that Global Prologs are executed before the AssemblyLine's own Prolog Hooks.

GUI (`ibmditk` or `ibmditk.bat`)

The term "TDI GUI" is sometimes used to refer to the Config Editor.

Hook

This is a *waypoint* in the built-in workflow of the AssemblyLine, or of a Connector or Function, where you can customize behavior by writing JavaScript. In a Connector, the Hooks available are also dependent on the mode setting.

HTML

Hyper**T**ext **M**arkup **L**anguage. a more or less standardized way of describing and formatting a page of text on the WordWide Web. Different manufacturers' interpretations of the standard are often the cause of Web Browser's different renderings of a given page.

HTTP

Hyper**T**ext **T**ransfer **P**rotocol. The protocol in use for the WorldWide Web, another protocol on top of TCP.

IEHS IBM Eclipse Help System. Used to host the TDI documentation locally. The documentation hosted by IBM in the Documentation Library also uses IEHS.

Initial Work Entry (IWE)
This is an Entry that is passed into an AssemblyLine by the process that called it (for example, an AssemblyLine Connector or Function, or by using script calls like `main.startAL()`. Note that the presence of an IWE will cause any Iterators in the Flow section to skip on this cycle.

Iterator
A Connector mode[10] that first creates a data result set (for example, by issuing a SQL SELECT statement, a LDAP search operation, opening a file for input, and so forth) and then returns one Entry at a time to the AL for processing. Iterators can reside in the AssemblyLine Feeds section where they drive data to Flow components. If they are placed in the Flow section then they still retrieve the next Entry from their result set for each AL cycle, but they do not *drive* AL cycling in this case.

IU Installation Unit. A term specific to Solution Install (SI). Each major component of the product is broken into separate IUs - for ease of maintenance, installation and updates.

Java Virtual Machine or JVM
The JVM IBM Tivoli Directory Integrator runs inside what is known as a Java Virtual Machine. It has its own memory management and is in most respects a Machine within the Machine.

Javadocs
A set of low-level API documentation, embedded in the product's source code and extracted by means of a special process during the product's build. In IBM Tivoli Directory Integrator the Javadocs can be viewed by selecting **Help>Low Level API** from the Config Editor.

JavaScript
The language you can use to fine tune the behavior of your AssemblyLines. TDI 6.1.1 uses the IBM JSEngine.

JMS **J**ava **M**essaging **S**ervice. A standard protocol used to perform guaranteed delivery of messages between two systems.

JNDI **J**ava **N**aming and **D**irectory **I**nterface. See "JNDI Connector", in the *IBM Tivoli Directory Integrator 6.1.1: Reference Guide*.

Link Criteria
Link Criteria represent the matching rules defined for a Connector in Update, Lookup or Delete, and they must result in a single entry match in the connected system; otherwise either an Not Found or Multiple Found exception occurs. Note that a Lookup Connector tied to a Loop is an efficient way of dealing with lookup operations where no match (or multiple matches) are expected.

LDAP **L**ightweight **D**irectory **A**ccess **P**rotocol. An easier way of accessing (using TCP) a name services directory than the older Directory Access Protocol. Used in for example querying the IBM Directory Server.

Memory Queue (MemQ)
The MemQ is a TDI object that lets you pass any type of Java object (like Entries) between AssemblyLines running on the same Server. This feature is usually accessed through the MemQueue Connector (or the deprecated Memory Queue FC). See also System Queue for more on how to pass data between running ALs.

Message Prefix
All error messages and Info messages in IBM Tivoli Directory Integrator are prefixed with a unique Message Prefix. The prefix assigned to TDI is **CTGDI**.

[10] Connectors running in Iterator mode are often referred to as "Iterators".

Mode Connectors have a mode setting that determines how this component will participate in AssemblyLine processing. In addition to the custom modes (implemented through Adapters) there is a set of standard modes:

- Iterator
- AddOnly
- Lookup
- Update
- Delete
- CallReply
- Server
- Delta

Dependent on the features provided by the underlying system or functionality built into the Connector, the list of modes supported by the different Connectors will vary. See ″Connectors″ in *IBM Tivoli Directory Integrator 6.1.1: Reference Guide* for more information about Connector modes.

Null Value Behavior
This term refers to how TDI will deal with Attribute Mappings that result in "null" values. Null Behavior configuration can be done for a Server by setting Global/Solution properties. These Server-level settings can be overridden for an Attribute Map by pressing the **Null** button in the button bar at the top of the map; or for a specific Attribute via the **Null** button in the Details Window for its mapping.

TDI lets you both configure what constitutes a "null" value situation (for example, missing values, empty string or a specific value) as well as how to handle this.

Op-Entry (Operation Entry)
An entry which contains information about the Operation for the currently executing AL. An Op-Entry persists its value over successive cycles for the same AL run and is available for scripting via the `task.getOpEntry()` method.

Parameter Substitution
A way of specifying patterns based on Java MessageFormat class - for simpler/quicker editing. Available in various places in TDI.

Parser TDI components used to interpret or generate the structure for a byte stream. Parsers are used by attaching them to a Connector that reads/writes byte streams, or to a Function component like the Parser FC which is used to parse data in the Work Entry.

Persistent Object Store
See System Store.

Persistent Parameter Store
See Property Store.

Prolog A set of Hooks that, if enabled, are run during the AssemblyLine Initialization phase. You can also define Global Prologs: Scripts that are run before either of the AL Prolog Hooks. Note that the "At Startup" initialization of components occurs between the two AL Prolog Hooks, which means that the Prolog Hooks of these components are all completed before the AssemblyLine Prolog - After Initialization Hook is called. See also Epilog.

Properties
This term refers to values maintained in a Property Store and used to configure AssemblyLine and Component settings at run-time[11].

[11] Note that an Entry object can also hold *properties* (in addition to Attributes and delta operation codes) and these can be accessed via the `getProperty()` and `setProperty()` methods of the Entry class.

Property Store

This is a feature for reading and writing all types of properties. This includes:

- Java-Properties, which are settings of the JVM.
- Global-Properties, TDI Server settings that are kept in a file called `global.properties` the "etc" folder of your installation directory.
- Solution-Properties, which typically override Global-Properties and are found in a file in your solution directory called `solution.properties`.
- System-Properties, for keeping custom property settings (uses the System Store).

In addition, you can define your own Property Stores using a Connector. The Property Store feature also lets you designate one of your Property-Stores as a *Password Store*, giving you automatic protection of sensitive configuration details.

Raw Connector

Deprecated term; this is now called the Connector Interface and refers to the part of an AL Connector that contains the logic needed to access a specific API, protocol or transport.

Relative Distinguished Name (RDN™)

In LDAP terms the name of an object that is unique relative to its siblings. RDNs have the form *attribute name=attribute value*.

```
cn=John Doe
```

Resource Library

A simple method for sharing AssemblyLines and components between Configs. In the Config Editor, the “Resources” navigator appears just below the Config Browser.

RMI **R**emote **M**ethod **I**nvocation; a way of making procedure or method calls on a remote system using a network communication channel. In TDI, used by the Remote API functionality.

Sandbox

The feature of the IBM Tivoli Directory Integrator that enables you to record AssemblyLine operations for later playback without any of the data sources being present. See ″Sandbox″ in *IBM Tivoli Directory Integrator 6.1.1: Users Guide*.

SAP Used to stand for "Systeme, Anwendungen, Produkte" (Systems, Applications, Products) but today, the abbreviation just stands for itself. A large, German provider of an integrated suite of ERP applications. Mostly known for its R/3 distributed ERP software suite, but also known for its mainframe-based R/2 software.

Script component (SC)

A Script is a block of JavaScript that is stored as a single component in TDI. The scripts library appears in the <tdi_installation> folder.Config Browser[12], Scripts can be dropped anywhere in the Flow section of an AssemblyLine.

Script Engine

The component that interprets the Java scripts written inside a TDI Config. The IBM jsEngine is used by TDI 6.1.1, which replaces Rhino from the previous releases.

Schema

The word "Schema", unfortunately, can mean different although related things, depending on context. In a relational database context, a schema is the collection of tables and objects a user has defined and owns (including content); and each table in a schema is described by a Data Definition. In an LDAP context, the Schema is the actual layout of the LDAP database, with its attributes and objects.

[12] In order to be used as Global Prologs (which are executed at the very start of Assemblyline Initialization) the Script must be in the scripts library folder and selected for inclusion in the Config tab of an AssemblyLine.

In addition, Connectors and Functions can have Input and Output schemas that represent the data model discovered in a connected system. Furthermore, an AssemblyLine Operation can have an Input and Output schema as well.

In a product like TDI, which with equal ease can access both relational databases as well as LDAP databases, the word Schema can therefore mean different things, depending on where it is used.

Script Connector
A Script Connector is a Connector where you write the *Interface* functionality yourself: It is empty in the sense that, in contrast to an already-existing Connector, the Script Connector does not have the base methods `getNextEntry()`, `findEntry()` and so forth implemented. Not to be confused with the Script Component.

Server (`ibmdisrv` or `ibmdisrv.bat`)
This is the part of the TDI product that is used to deploy and execute Configs.

Server (mode)
This is a Connector mode used for providing a request/response service (like an HTTP server). This mode also provides an AssemblyLine Pool feature to enable support for more connections/traffic.

Solution Directory
The directory in which you store your Config files, CloudScape databases, properties files, keystores and so forth. The solution directory is selected when you install TDI, and the filepaths used in your solution can be relative to this folder. The solution directory can be explicitly specified when you start the Config Editor or Server using the **-s** commandline option. Note that the counterpart of `global.properties` is kept in this folder and called `solution.properties`—unless, of course, your solution directory is the same as your installation directory.

SI Solution Installer. A common IBM utility for installation of many IBM products. The TDI installer is one such product.

SSL Secure Socket Layer; a protocol used in Internet communications to encrypt data such that if someone where to eavesdrop on the packets going back and forth he would not be able to see what the packets contain. The protocol was invented by Netscape; and you can see if a Web page uses the SSL protocol to talk to the Web server if it has the 'https//' prefix instead of 'http'. SSL is by no means limited to Web pages; in fact, TDI uses it (if configured that way) to talk between different TDI Servers and AssemblyLines if network access is called for.

State Defines the *level of participation* for an AssemblyLine component. It can be in either *Enabled* State, which means it will participate in AL processing, or *Disabled* in which case the component is not used in any way.

Connectors and Functions can be set to a third State: *Passive*. Passive State causes the component to be initialized and closed during the Assemblyline Initialization and Shutdown phases, but never used during AL cycling. However, you can drive these components manually through script calls.

System Queue
A built-in queue infrastructure to facilitate the guaranteed delivery of messages between AssemblyLines, even running on different TDI Servers. By default, the System Queue uses the bundled MQe (WebSphere MQ Everyplace), but can be configured to leverage other JMS-compliant messaging systems. TDI provides a SystemQueue Connector to help you leverage this feature.

New with TDI 6.1, System Queue is a built-in queue infrastructure to facilitate communications between ALs, in much the same way that the MemQ feature (and components) do. The big difference is that the System Queue enables data transfer across multiple TDI Servers. By default, the System Queue uses the bundled MQe (WebSphere MQ Everyplace), but can be configured to leverage other JMS-compliant queuing systems.

For more information about the System Queue and how to enable it, see the "System Queue" chapter in the *IBM Tivoli Directory Integrator 6.1.1: Administrator Guide*.

System Store
Called the Persistent Object Store, or POS in older TDI versions, the System Store is a relational database used to store state information, like Delta Tables (used by the Delta Engine) or Iterator state for Change Detection Connectors. It also provides the User Property Store which is accessible through the `system.setPersistentObject()`, `system.getPersistentObject()` and `system.deletePersistentObject()` methods. In the current implementation, the IBM DB2 for Java product (also known as **CloudScape**) is used. See http://www-3.ibm.com/software/data/cloudscape for more details.

Task By convention, all threads (AssemblyLines, EventHandlers and so forth) are referred to as *tasks* and are accessible from script code via the pre-registered **task** variable.

Task Call Block
A Java structure used to pass parameters to and from AssemblyLines. Often referred to by its abbreviation: **TCB**.

TCP **T**ransmission **C**ontrol **P**rotocol, a level 4 (transmission integrity) protocol usually seen in combination with its layer 3 (routing) Internet Protocol as in TCP/IP. A stack of protocols designed to achieve a standardized way of communicating across a network, be it local (as in on the premises) or over long distances. Originally invented and specified by DARPA, the (US) Defense Advanced Research Projects Agency. Successor to ARPANET, which was a network of a (small) number of universities and the US Department of Desfense, the civil side of which was managed by the Stanford Research Institute (SRI). TCP is related to UDP.

TDI Unofficial monicker for this product, IBM **T**ivoli **D**irectory **I**ntegrator.

TMS XML
Tivoli Message Standard XML. A Tivoli standardized way of formatting messages. Each message is prefixed by a unique TMS code, which can be looked up in the Message Guide for explanation and user response. If the code ends in ″E″ - it indicates an Error, ″W″ indicates a warning and ″I″ indicates an Information message. All Tivoli messages issued by TDI start with this product's unique identifier, which is "CTGDI".

Tombstone
A record or trace showing that an AssemblyLine has terminated. Configured through the Tombstone Manager in the CE. The trace includes a timestamp and the AL exit status. The new Tombstone Manager creates a tombstone for each AssemblyLine as it terminates.

TWiki TWiki as a piece of software is a flexible and easy to use enterprise collaboration system. Its structure is similar to the WikiPedia, except that is not linked into that. It is rather meant as an independent community resource for a group of people with common interest. There is one for IBM Tivoli Directory Integrator as well, at http://www.tdi-users.org.

Note: The TWiki site is a volunteer effort, and is not an official Tivoli support forum. If you need immediate assistance contact your local Tivoli support organization.

Update
One of the standard Connector modes. Update mode causes the Connector to first perform a lookup for the entry you want to update[13], and if found it modifies this entry. If no match is found then a new entry is added instead. See also Computed Changes.

[13] Data is read into both the `conn` and `current` Entry objects. After the Output Map, the contents of `conn` are now the Attributes to be written. The original entry data is still available in `current`.

UDP **U**ser **D**atagram **P**rotocol. A protocol used on top of the Internet Protocol (IP) which, unlike TCP does **not** guarantee that the packet of data sent with it reaches the other end. Also see TCP.

URL **U**nified **R**esource **L**ocator. A way of defining where a resource is, be it a fileserver or a HTML page on the WordlWide Web.

User Property Store
See Property Stores in the *IBM Tivoli Directory Integrator 6.1.1: Users Guide*.

Value (data values and types)
See Entries, and Attribute.

WikiPedia
A Web-based world-wide encyclopedia, where (registered) users can add articles or pictures, edit them, browse them, search for applicable content, and so forth For TDI there is one that similar in functionality but not linked into the WikiPedia, a "TWiki" at http://www.tdi-users.org. The TWiki is a groupware product.

Work Entry
An Entry object that is used by the AssemblyLine to carry data from component to component[14]. This object can be accessed in script code via the pre-defined variable `work`. The Work Entry is typically built by a Server or Iterator mode Connector in the Feeds section before being passed to the AL Flow section. You can also have an Initial Work Entry (IWE) passed in if the AL was called from another process; or you can create it in the Prolog by using `task.setWork()`:

```
init_work = system.newEntry(); // Create a new Entry object
init_work.setAttribute("uid", "cchateauvieux"); // populate it
task.setWork(init_work); // make it known as "work" to the Connectors
```

Note that an Iterator in the Feeds section will not return any data if the Work Entry is already defined at this point in the AL. So if an IWE is passed into an AssemblyLine, any Iterators in the Feeds section will simply pass control to the next component in line. It is also the reason why multiple Iterators in the Feeds section run sequentially, one starting up when the previous one reaches End-of-Data.

XML The **X**tensible **M**arkup **L**anguage. A general purpose markup language (See also HTML) for *creating* special-purpose markup languages, and also capable of describing many types of data IBM Tivoli Directory Integrator uses XML to store Config files.

[14] Note that the "Work Entry" window shown in the Config Editor is actually a list of all Attributes that appear in Input Maps or in the Loop Attribute field of Loops in the AssemblyLine.

Appendix B. Notices

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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

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

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

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

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
Department MU5A46
11301 Burnet Road

Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

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

COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Third-Party Statements

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Trademarks

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

IBM
Notes
OS/390®
Tivoli
pSeries®
Domino®
AIX®
DB2
iNotes
Lotus
WebSphere
Cloudscape

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

Microsoft, Windows NT and Windows are registered trademarks of Microsoft Corporation.

Intel™ is a trademark of Intel Corporation in the United States, other countries, or both.

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

Linux is a trademark of Linus Torvalds in the U.S., other countries, or both.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

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

Printed in USA

SC32-2567-01