

Directory Integrator
Version 7.1.1

Installation and Administrator Guide



Directory Integrator
Version 7.1.1

Installation and Administrator Guide



Note

Note: Before using this information and the product it supports, read the general information under Appendix D, "Notices," on page 363.

Edition notice

This edition applies to version 7.1.1 of the IBM Tivoli Directory Integrator and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 2003, 2012.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

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.

The reader should be familiar with the concepts and the administration of the systems that the developed solutions 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
- Windows and UNIX operating systems
- Security management
- Internet protocols, including HTTP, HTTPS and TCP/IP
- Lightweight Directory Access Protocol (LDAP) and directory services
- Supported user registry
- Authentication and authorization
- SAP ABAP Application Server

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 Tivoli Directory Integrator library are:

IBM Tivoli Directory Integrator V7.1.1 Getting Started

A brief tutorial and introduction to Tivoli Directory Integrator 7.1.1. Includes examples to create interaction and hands-on learning of IBM Tivoli Directory Integrator.

IBM Tivoli Directory Integrator V7.1.1 Installation and Administrator Guide

Includes complete information about installing, migrating from a previous version, configuring the logging functionality, and the security model underlying the Remote Server API of IBM Tivoli Directory Integrator. Contains information on how to deploy and manage solutions.

IBM Tivoli Directory Integrator V7.1.1 Users Guide

Contains information about using IBM Tivoli Directory Integrator 7.1.1. Contains instructions for

designing solutions using the Tivoli Directory Integrator designer tool (**ibmditk**) or running the ready-made solutions from the command line (**ibmdisrv**). Also provides information about interfaces, concepts and AssemblyLine creation.

IBM Tivoli Directory Integrator V7.1.1 Reference Guide

Contains detailed information about the individual components of IBM Tivoli Directory Integrator 7.1.1: Connectors, Function Components, Parsers and so forth – the building blocks of the AssemblyLine.

IBM Tivoli Directory Integrator V7.1.1 Problem Determination Guide

Provides information about IBM Tivoli Directory Integrator 7.1.1 tools, resources, and techniques that can aid in the identification and resolution of problems.

IBM Tivoli Directory Integrator V7.1.1 Messages Guide

Provides a list of all informational, warning and error messages associated with the IBM Tivoli Directory Integrator 7.1.1.

IBM Tivoli Directory Integrator V7.1.1 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 Directory Server Password Synchronizer, IBM Tivoli Directory Server Password Synchronizer, Domino Password Synchronizer and Password Synchronizer for UNIX and Linux. Also provides configuration instructions for the LDAP Password Store and JMS Password Store.

IBM Tivoli Directory Integrator V7.1.1 Release Notes

Describes new features and late-breaking information about IBM Tivoli Directory Integrator 7.1.1 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 7.1.1 uses the JNDI client from Oracle. For information about the JNDI client, refer to the *Java Naming and Directory Interface™ Specification* at: <http://download.oracle.com/javase/6/docs/technotes/guides/jndi/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, at: <http://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 **Fit to page** 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 Tivoli Directory Integrator 7.1.1, 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 Tivoli Directory Integrator 7.1.1:

- 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 Microsoft Windows navigation keys for common Windows actions such as access to the File menu, copy, paste, and delete. Actions that are unique to Tivoli Directory Integrator use Tivoli Directory Integrator keyboard shortcuts. Keyboard shortcuts have been provided wherever needed for all actions.

Interface information

The accessibility features of the user interface and documentation include:

- Steps for changing fonts, colors, and contrast settings in the Configuration Editor:
 1. Type Alt-W to access the Configuration Editor **Window** menu. Using the downward arrow, select **Preferences...** and press Enter.
 2. Under the **Appearance** tab, select **Colors and Fonts** settings to change the fonts for any of the functional areas in the Configuration Editor.
 3. Under **View and Editor Folders**, select the colors for the Configuration Editor, and by selecting colors, you can also change the contrast.
- Steps for customizing keyboard shortcuts, specific to IBM Tivoli Directory Integrator:
 1. Type Alt-W to access the Configuration Editor **Window** menu. Using the downward arrow, select **Preferences...**.
 2. Using the downward arrow, select the General category; right arrow to open this, and type downward arrow until you reach the entry **Keys**.
Underneath the **Scheme** selector, there is a field, the contents of which say "type filter text." Type **tivoli directory integrator** in the filter text field. All specific Tivoli Directory Integrator shortcuts are now shown.
 3. Assign a keybinding to any Tivoli Directory Integrator command of your choosing.
 4. Click **Apply** to make the change permanent.

The Configuration Editor is a specialized instance of an Eclipse workbench. More detailed information about accessibility features of applications built using Eclipse can be found at <http://help.eclipse.org/help33/topic/org.eclipse.platform.doc.user/concepts/accessibility/accessmain.htm>

- The 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 IA 2010 SP1 installer technology.

The IBM Tivoli Directory Integrator 7.1.1 installer has accessibility features that are independent from the product. The installer supports 3 UI modes:

GUI Keyboard-only operation is supported in GUI mode, and the use of a screen reader is possible. In order to get the most from a screen reader, you should use the Java Access Bridge and launch the installer with a Java access Bridge enabled JVM, for example:

```
install_tdiv711_win_x86.exe LAX_VM "Java_DIR/jre/bin/java.exe"
```

The JVM used should be a Java 6 JRE.

Console

In console mode, keyboard-only operation is supported and all displays and user options are displayed as text that can be easily read by screen readers. Console mode is the suggested install method for accessibility.

Silent In silent mode, user responses are given through a response file, and no user interaction is required.

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/IBMDirectoryIntegrator.html>

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.

A list of most requested documents as well as those identified as valuable in helping answer your questions related to IBM Tivoli Directory Integrator can be found at <http://www-01.ibm.com/support/docview.wss?rs=697&uid=swg27009673>.

Contents

Preface	iii
Who should read this book	iii
Publications	iii
IBM Tivoli Directory Integrator library	iii
Related publications	iv
Accessing publications online	iv
Accessibility	v
Accessibility features	v
Keyboard navigation	v
Interface information	v
Vendor software	v
Related accessibility information	vi
Contacting IBM Software support	vi
Chapter 1. Introduction	1
IBM Tivoli Directory Integrator 7.1.1 Editions	1
Chapter 2. Installation instructions for IBM Tivoli Directory Integrator	3
Before you install	3
Disk space requirements	3
Memory requirements	3
Platform requirements	3
Components in IBM Tivoli Directory Integrator	4
Other requirements	7
Installing IBM Tivoli Directory Integrator	8
Launching the appropriate installer	9
Using the platform-specific TDI installer	11
Installing using the graphical installer	13
Installing using the command line	42
Temporary file space usage during installation	44
Performing a silent install	44
Post-installation steps	44
Installing local Help files	46
Deploying AMC to a custom ISC SE or Tivoli Integrated Portal (ISC embedded)	47
Installing or Updating using the Eclipse Update Manager	48
Post-installation steps	50
Uninstalling	51
Launching the uninstaller	51
Performing a silent uninstallation	52
Default installation locations	52
Chapter 3. Update Installer	53
The .registry file	55
Installing fixes	56
Rollback	57
Troubleshooting	57
Chapter 4. Supported platforms	59
Virtualization support	61
Chapter 5. Migrating	63
Migrate files to a different location	63
Which files do not need to be modified to be used in another location?	63
Which files need to be modified before they can be used in another location?	64
Which files should not be used in another location under normal circumstances?	64
Migrating files that contain encrypted data	65
Migrate files to a newer version	65
Installer-assisted migration	65
Tool-assisted migration	66
Manual migration	66
Backing up important data	76
Migrating AMC 7.x configuration settings to another AMC deployment	79
Converting from EventHandlers to corresponding AssemblyLines	79
TCP Server Connector	80
Mailbox Connector	80
JMX Connector	81
SNMP Server Connector	81
IBM Directory Server Changelog Connector	81
HTTP Server Connector	82
LDAP Server Connector	82
Sun Directory Change Detection Connector	83
Active Directory Change Detection Connector	83
z/OS LDAP Changelog Connector	84
DSMLv2SOAPServerConnector	85
Migrating BTree tables and BTree Connector to System Store	85
Migrating Cloudscape database to Derby	86
Migrating global and solution properties files using migration tool	87
Migrating Password plug-ins properties files using migration tool	88
Chapter 6. Security and TDI	89
Introduction	89
Manage keys, certificates and keystores	89
Background	89
List the contents of a keystore	91
Create keys	91
Secure Sockets Layer (SSL) Support	94
Server SSL configuration of TDI components	95
Client SSL configuration of TDI components	96
SSL client authentication	96
IBM Tivoli Directory Integrator and Microsoft Active Directory SSL configuration	97
Summary of properties for enabling SSL and PKCS#11 support	98
SSL example	99
Remote Server API	100
Introduction	100
Configuring the Server API	101
Server API access options	104

Server API SSL remote access	105
Server API authentication	106
Server API Authorization	114
Server Audit Capabilities	119
Tivoli Directory Integrator Server Instance Security	121
Stash File	122
Server Security Modes	122
Working with encrypted TDI configuration files	123
Standard TDI encryption of global.properties or solution.properties.	125
Encryption of properties in external property files	125
The TDI Encryption utility	126
TDI System Store Security	127
Miscellaneous Config File features	129
The "password" configuration parameter type	129
Component Password Protection	129
Protecting attributes from being printed in clear text during tracing	130
Encryption of TDI Server Hooks	130
Remote Configuration Editor and SSL	130
Using the Remote Configuration Editor	131
Summary of configuration files and properties dealing with security	131
Web Admin Console Security	134
Miscellaneous security aspects	134
HTTP Basic Authentication	134
Lotus Domino SSL specifics	135
Certificates for the TDI Web service Suite	135
MQe authentication with mini-certificates	135
Chapter 7. Reconnect Rule Engine	137
Introduction	137
Reconnect Rules	137
User-defined rules configuration	139
Examples	139
Exception considerations	140
General reconnect configuration	140
Chapter 8. System Queue	143
System Queue Configuration	143
Apache ActiveMQ parameters	143
WebSphere MQ parameters	145
WebSphere MQ parameters	145
Microbroker parameters	146
JMSScript Driver parameters	146
System Queue Configuration Example	148
Security and Authentication	148
MQe Configuration Utility	149
Authentication of MQe messages to provide MQe Queue Security	149
Support for DNS names in the configuration of the MQe Queue	150
Configuration of High Availability for MQe transport of password changes	150
Providing remote configuration capabilities in the MQe Configuration Utility	151
Chapter 9. Encryption and FIPS mode	153
Configuring Tivoli Directory Integrator to run FIPS mode	153
Symmetric cipher support	153
Configuring SSL and PKI certificates	161
Encrypting and decrypting using CryptoUtils	162
Working with certificates	162
Using cryptographic keys located on hardware devices	163
Using IBMPCKS11 to access devices and to store SSL keys and certificates	164
Enabling or disabling padding	164
Maintaining encryption artifacts – keys, certificates, keystores, encrypted files	164
Changed encryption key	165
Changed password for encryption key or keystore	165
Expired encryption certificate	165
Chapter 10. Configuring the TDI Server API.	167
Server ID	167
Exception for password protected Configs	167
Server RMI	167
Config load time-out interval	168
Chapter 11. Properties	169
Working with properties	169
Migrating using properties and the tdimiggl tool	170
Global properties	170
Solution properties	170
Java properties	170
System properties	172
Chapter 12. System Store	173
Property stores	173
Password Store	173
User property stores	174
Third-party RDBMS as System Store	174
Oracle	175
MS SQL Server	175
IBM DB2 for z/OS	176
DB2 for other OS	177
IBM SolidDB	177
Using Derby to hold your System Store	177
Configuring Derby Instances	178
Starting Derby in networked mode	179
Enabling user authentication in System Store	179
Create statements for System Store tables	179
Backing up Derby databases	180
Troubleshooting Derby issues	181
Pre-6.0 (properties file) configuration of Cloudscape	182
See also	184
Chapter 13. Command-line options	185
Configuration Editor	185
Server	186
Command Line Interface – tdisrvctl utility	189
Command Line Reference	189

Chapter 14. Logging and debugging	201
Script-based logging	202
Logging using the default Log4J class	202
Log Levels and Log Level control	206
Log4J default parameters	206
Creating your own log strategies	207
Chapter 15. Tracing and FFDC	209
Tracing Enhancements	209
Understanding Tracing	209
Configuring Tracing	210
Setting trace levels dynamically	210
Useful JLOG parameters.	211
Chapter 16. Administration and Monitoring	213
Installation and Configuration	213
Deploying AMC into the Integrated Solutions Console (ISC)	213
Starting the Administration and Monitoring Console and Action Manager and logging in	214
Enabling AMC	215
AMC Logs	216
Compatibility with previous versions of Tivoli Directory Integrator	217
AMC in the Integrated Solutions Console	218
Console user authority	218
Action Manager	219
Enabling Action Manager	223
Action Manager status in real time	224
AMC force trigger for a given rule	225
AMC and Action Manager security	225
Introduction	225
AMC and SSL	226
AMC and remote TDI server	227
AMC and role management	227
AMC and passwords	228
AMC and encrypted configs	228
Administration and Monitoring Console User Interface	229
Log in and logout of the console	229
AMC Console Layout	230
Logging off the console	231
Using AMC tables	231
Servers	233
Console Properties	234
Solution Views	235
Monitor Status and Action Manager	239
Property Stores	249
Log Management	250
Preferred Solution Views	251
AMC and AM Command line utilities	251
Example walkthrough of creating a Solution View and Rules	256
Chapter 17. Touchpoint Server	263
Touchpoint concepts	263
Touchpoint Server	263
Touchpoint Provider	263
Touchpoint Type	264
Touchpoint Instance	265
Touchpoint Template	267
Resource Persistence	271
Touchpoint Schema	272
Touchpoint Server communication protocol	272
Touchpoint Configuration	276
Touchpoint Instance communication protocol	278
Touchpoint Status Entry schema	280
Property sheet definitions	281
XML Schema locations	282
Error flows	282
Configuration	283
Authentication	285
Examples.	285
Shipped example	285
Example steps for creating a Touchpoint Instance using a JDBC Connector.	286
Chapter 18. Tombstone Manager	291
Introduction	291
Configuring Tombstones.	291
Configuration Editor Configuration screen.	291
AssemblyLine Configuration screen	293
The Tombstone Manager	294
Chapter 19. Multiple TDI services	297
IBM Tivoli Directory Integrator as Windows Service	297
Introduction	297
Installing and uninstalling the service	297
Starting and stopping the service.	298
Logging	298
Configuring the service	298
IBM Tivoli Directory Integrator as Linux/UNIX Service	300
Deployment methods.	300
Tailoring /etc/inittab.	300
IBM Tivoli Directory Integrator as z/OS Service	302
USS process	302
Normal z/OS started task	302
IBM Tivoli Directory Integrator as i5/OS Service	303
Command line support	304
Chapter 20. z/OS environment Support	305
Post install configuration	305
Using MQe for system queue	305
Default encoding different than IBM-1047	305
JDK 5.0 not located at /usr/lpp/java/J5.0.	306
Running Tivoli Directory Integrator	306
Reading License Files.	306
Using the Remote Configuration Editor on z/OS	307
Handling configuration and properties files	307
Using ASCII mode	308
Configuring the TDI task to log to its SYSOUT	309
Appendix A. Dictionary of terms	311
IBM Tivoli Directory Integrator terms	311
Appendix B. Example Property files	325
Log4J.properties	325

jlog.properties	327	Monitoring TDI data	345
derby.properties	328	Send custom notifications to ITM	357
global.properties	328	Limitations	357
Appendix C. Monitoring with external tools	335	Monitoring TDI using OMNIbus	358
Monitoring TDI with ITM	335	Introduction	358
Short presentation of the ITM architecture	335	Configuring the EIF probe props file	358
Importing an existing Agent configuration in ITM Agent Builder 6.2	336	Determine the severity for the events	358
Creating a Tivoli Directory Integrator agent for ITM using ITM Agent Builder 6.2	336	Working with the EventPropertyFile.properties file	359
Generating the ITM Agent	344	Send custom notifications to OMNIbus	360
Configuring the ITM Agent	345		
Appendix D. Notices	363		
Trademarks	365		

Chapter 1. Introduction

For an overview of the general concepts of the IBM Tivoli Directory Integrator 7.1.1, refer to "IBM Tivoli Directory Integrator concepts," in *IBM Tivoli Directory Integrator V7.1.1 Users Guide*.

For more detailed information about IBM Tivoli Directory Integrator 7.1.1 concepts, see *IBM Tivoli Directory Integrator V7.1.1 Reference Guide*.

IBM Tivoli Directory Integrator 7.1.1 Editions

The IBM Tivoli Directory Integrator 7.1.1 exists in two different editions (for which different licensing agreements apply):

Identity Edition

The Identity Edition contains the full set of Connectors, Parsers, Function Components, Password Interceptor Plug-ins and miscellaneous other components. Licensing is done on a per-user basis.

General Purpose Edition

The General Purpose Edition differs from the Identity Edition in that some components have been removed. Those removed are the specific Identity Management components as listed below:

- Windows Users and Groups Connector
- z/OS Changelog Connector
- IDS Changelog Connector
- Sun Directory Changelog Connector
- Active Directory Change Detection Connector
- ITIM DSMLv2 Connector
- TAM Connector
- JMS Password Store Connector
- User Registry Connector for SAP R/3
- LDIF Parser
- SPMLv2 Parser
- Password Interceptor Plug-ins

Examples applying to these components have been removed as well.

Licensing for this edition is done on a per-processor basis.

Chapter 2. Installation instructions for IBM Tivoli Directory Integrator

Before you install

The Tivoli Directory Integrator 7.1.1 installer uses the InstallAnywhere 2010 technology. Before you install, read the following sections and make sure your system meets the minimum requirements.

Disk space requirements

The IBM Tivoli Directory Integrator 7.1.1 installer requires 450 MB of temporary disk space during installation, and additionally the following amount of disk space for the TDI components that remain on the computer after installation:

- Windows 32-bit: 629 MB
- Windows 64-bit: 665 MB
- Linux 32-bit: 577 MB
- Linux 64-bit: 595 MB
- Linux PPC: 410 MB
- zLinux s390: 421 MB
- AIX: 609 MB
- AIX PPC 64 bit: 430MB
- Solaris SPARC: 652 MB
- Solaris Opteron: 492 MB
- HP-UX Itanium: 788 MB
- i5OS: 350 MB

The precise amount of required disk space depends on the components you choose to install; the amounts listed above are applicable for a full Custom option installation. 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. A default installation generally requires 30MB less than the numbers listed above.

Memory requirements

The IBM Tivoli Directory Integrator 7.1.1 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 Tivoli Directory Integrator component.

Memory requirements for a Typical installation: 484 MB

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

Platform requirements

See Chapter 4, “Supported platforms,” on page 59

Components in IBM Tivoli Directory Integrator

With some exceptions, the following components are available and selectable for installation as part of IBM Tivoli Directory Integrator 7.1.1:

Runtime Server

A rules engine used to deploy and run Tivoli Directory Integrator integration solutions.

- Disk space requirements: 48 MB.
- Memory requirements: Each server instance requires at least 256 MB. NOTE: More RAM may be required depending on the size and complexity of the solution being created.

Configuration Editor

A development environment for creating, debugging and enhancing TDI integration solutions.

Note: IBM Tivoli Directory Integrator does not support the Configuration Editor (CE) on the following operating systems:

- HP-UX Integrity*
- Solaris Opteron
- z/OS
- i5/OS
- Linux PPC
- Linux 390

* On HP-UX Integrity, you have the option of installing the Tivoli Directory Integrator Eclipse CE plug-ins into an existing Eclipse Workbench; see below for more details. If you choose not to do that, and for the other platforms, see "Using the Remote Configuration Editor" on page 131 and "Using the Remote Configuration Editor on z/OS" on page 307.

Disk space requirements for the Configuration Editor for each supported operating system:

- Microsoft Windows: 139 MB
- Linux: 139 MB
- AIX 139 MB
- Solaris 138 MB

Memory requirements for CE on each supported operating system: 128 MB.

Configuration Editor Update Site (Eclipse update site for CE)

Use the CE Update Site folder to install the Tivoli Directory Integrator Configuration Editor into an existing Eclipse installation. Use the Eclipse software update tool and use this folder as a local update site. The CE update site is only supported for deployment on Eclipse 3.5.1.

Note: IBM Tivoli Directory Integrator does not support the Configuration Editor Update Site on the following operating systems:

- Solaris Opteron
- z/OS
- i5/OS
- Linux PPC
- Linux 390

See "Using the Remote Configuration Editor" on page 131 and "Using the Remote Configuration Editor on z/OS" on page 307.

CE Update Site requirements are:

- Disk space requirements: 6 MB
- Memory requirements: Not applicable

Java API documentation

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

- Disk space requirements: 48 MB
- Memory requirements: Not applicable

Examples

A series of short, illustrative example Configs that highlight specific Tivoli Directory Integrator features or components.

- Disk space requirements: 3 MB
- Memory requirements: Not applicable

Help system, v3.3.1 (Host TDI help locally. The default is online.)

You can install an IBM User Interface Help System built on Eclipse (formerly known as IBM Eclipse Help System, or IEHS) locally as an alternative to using the global online help service. This option requires manual download and deployment of the Tivoli Directory Integrator help files after installation.

Disk space requirements by platform:

- Windows: 24 MB
- Linux: 18 MB
- AIX: 18 MB
- Solaris: 18 MB
- HP-UX: 18 MB
- i5/OS: 18 MB

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

Note: You must 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 installation instructions documented in "Installing local Help files" on page 46.

embedded Web platform (includes Integrated Solutions Console SE) v8.1.0.3

Tivoli Directory Integrator includes an embedded lightweight Web server platform, sometimes referred to as "LWI". This server platform is based on the Eclipse and Open Services Gateway Initiative (OSGI) architecture and supports running web applications and Web services. The runtime provides a secure infrastructure with a small footprint and minimal configuration. The embedded Web platform includes Integrated Solution Console SE, which is used as the default alternative for deploying AMC on an existing ISC installation. The embedded Web platform provides an OSGI based lightweight infrastructure for hosting Web applications and Web services with the following characteristics:

- Minimal footprint
- Minimal configuration
- Compatibility with OSGI based ISC

AMC installation into the embedded Web platform requires at least 94 MB or more on each of the following supported operating systems:

- Windows
- Linux
- AIX
- Solaris
- HP-UX

Memory requirements: a minimum of 512 MB is recommended.

Note: For the i5/OS® platform, the Integrated Web Application Server must already be installed on the target computer. See “Installing IBM Tivoli Directory Integrator on i5/OS” on page 12 for details.

AMC: Administration and Monitoring Console

A browser-based application for monitoring and managing running Tivoli Directory Integrator Servers. AMC runs in the Integrated Solutions Console (ISC). In previous releases, AMC was a servlet application that was deployed into an embedded or existing instance of WebSphere Application Server (WAS).

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

Tivoli Directory Integrator 7.1.1 supports ISC SE 7.2.0.2 and Tivoli Integrated Portal 2.1 (ISC embedded).

Additional components automatically installed that are not selectable:

JRE (Java Runtime Environment) 6.0 SR9

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

Note: The JRE used for any of the installed Tivoli Directory Integrator packages is independent of any system-wide JRE or JDK you may have installed on your system. The JRE is installed no matter what features are selected. The uninstaller requires the JRE, so it is always installed.

Disk space requirements by platform:

- Windows: 120 MB
- Linux: 91 MB
- AIX: 94 MB
- Solaris: 149 MB
- HP: 245 MB

Memory requirements: Not applicable

Password Synchronization Plug-ins

All supported platforms: 8 MB

Miscellaneous

Contains the License Package, the Uninstaller, the Update Installer and Tivoli Directory Integrator overhead.

The Tivoli Directory Integrator 7.1.1 License Package contains the license files for Tivoli Directory Integrator.

Disk space requirements by platform:

- Windows: 20 MB
- Linux: 20 MB
- AIX: 20 MB
- Solaris 19 MB
- HP-UX: 20 MB
- i5/OS: 20 MB

Memory requirements: Not applicable.

Other requirements

Root or Administrator Privileges

Note the following differences when installing Tivoli Directory Integrator with administrator as opposed to non-administrator privileges:

- Anyone installing Tivoli Directory Integrator must have write privileges when installing to the specified installation location.
- Non-administrator users have different Configuration Editor shortcuts from administrative users.
- Users who do not have administrator privileges when installing Tivoli Directory Integrator do not see the "Register AMC as a Service" and "Register Server as System Service" windows.
- Once Tivoli Directory Integrator is installed using one particular non-root user ID, that same user ID must be used to carry out any further maintenance on that installation, like un-installation or migration to newer versions.

Security Enhanced (SELinux)

RedHat Linux (RHEL) has a security feature known as Security Enhanced Linux or SELinux. SELinux provides security that protects the host from certain types of malicious attacks. A less secure version of SELinux was included in RHEL version 4.0 and was disabled by default, but RHEL version 5.0 defaults SELinux to enabled. The RHEL 5.0 SELinux default settings have been known to prevent Java from running properly. If you try to run the RHEL 5.0 Tivoli Directory Integrator installer, an error resembling the following output may display:

```
# ./install_tdiv711_linux_x86.bin  
  
Initializing Wizard.....  
Verifying JVM...  
  
No Java Runtime Environment (JRE) was found on this system.
```

The reason for this error is that the Java Runtime Environment (JRE) that InstallAnywhere 2010 extracts to the /tmp directory does not have the required permissions to run. To avoid this error:

1. Disable SELinux: `setenforce 0`.
2. Run the Tivoli Directory Integrator installer.
3. Enable SELinux again: `setenforce 1`.

You can also edit the /etc/selinux/config configuration file to enable and disable SELinux. The default settings for the /etc/selinux/config file resemble the following lines:

```
# This file controls the state of SELinux on the system.  
# SELINUX= can take one of these three values:  
# enforcing - SELinux security policy is enforced.  
# permissive - SELinux prints warnings instead of enforcing.  
# disabled - SELinux is fully disabled.  
SELINUX=enforcing  
# SELINUXTYPE= type of policy in use. Possible values are:  
# targeted - Only targeted network daemons are protected.  
# strict - Full SELinux protection.  
SELINUXTYPE=targeted
```

Modifying SELINUX to either SELINUX=permissive or SELINUX=disabled allows the Tivoli Directory Integrator installer to run. However, both modifications of the SELINUX property, to either SELINUX=permissive or to SELINUX=disabled, affect the level of security for the host.

The Tivoli Directory Integrator installer uses a JRE located at `install_dir/jvm` that cannot run with the SELinux default settings. The installer makes a best effort to avoid the problems with the SELinux default settings by trying to change the Tivoli Directory Integrator JRE security permissions that are blocking the

installer. The Tivoli Directory Integrator installer issues a command that changes the security permissions for the Tivoli Directory Integrator JRE so that it can run. The Tivoli Directory Integrator installer issues the following command:

```
chcon -R -t textrel_shlib_t install_dir/jvm/jre
```

Note: If the installer cannot issue the chcon command, or if there is an error when issuing the command, you must edit the permissions manually.

Errors that resemble the following output indicate that the chcon command did not work:

```
[root@dyn9-37-225-164 V7.1.1]# ./ibmdisrv  
Failed to find VM – aborting  
  
[root@dyn9-37-225-164 V7.1.1]# ./ibmditk  
Failed to find VM – aborting  
  
[root@dyn9-37-225-164 V7.1.1]# bin/amc/start_tdiamc.sh  
Failed to find VM - aborting
```

Authentication of AMC on Unix/Linux

On some UNIX platforms (for example, SLES 10) the Administration and Monitoring Console (AMC) in ISE SE fails consistently to authenticate users, even when correct credentials are specified. Such behavior is observed when AMC is run as a non-root user and the operating system uses a password database (for example, a /etc/shadow file). For more information on this issue, and for a workaround see "Authentication failure on UNIX when LWI runs as non-root user" in *IBM Tivoli Directory Integrator V7.1.1 Problem Determination Guide*.

Installing IBM Tivoli Directory Integrator

The Tivoli Directory Integrator installer allows you to install Tivoli Directory Integrator 7.1.1 in its entirety, only those Tivoli Directory Integrator components that you need, upgrade a previous version of Tivoli Directory Integrator (versions 6.0, 6.1, 6.1.1, 7.0, or 7.1), or add features to an existing Tivoli Directory Integrator 7.1.1 installation.

Note: IBM Tivoli Directory Integrator does not support the Configuration Editor (CE) on the following operating systems:

- HP-UX Integrity*
- Solaris Opteron
- z/OS
- i5/OS
- Linux PPC
- Linux 390

* On HP-UX Integrity, you have the option of installing the CE Plug-ins into an existing Eclipse Workbench; see "Platform requirements" on page 3. If you choose not to do that, and on the other platforms, see "Using the Remote Configuration Editor" on page 131 and "Using the Remote Configuration Editor on z/OS" on page 307 for more information on using the product without a locally-installed Configuration Editor.

Installing Tivoli Directory Integrator 7.1.1 uninstalls a previous version; the uninstallation does not remove any files that the user has created. User created files are still available after the new installation completes. Configuration files such as `global.properties` and `am_config.properties` are migrated to Tivoli Directory Integrator 7.1.1, keeping any custom configuration changes that have been made. The Tivoli Directory Integrator 7.1.1 installation continues to use the features available in previous versions of Tivoli Directory Integrator:

- Administration and Monitoring Console (AMC)
- Configuration Editor (CE)

- Examples
- IBM User Interface Help System built on Eclipse
- Java API Documentation
- Runtime Server

Note: For the remainder of this *IBM Tivoli Directory Integrator V7.1.1 Installation and Administrator Guide*, the variable *TDI_install_dir* refers to the installation directory location chosen by the user on the **Destination Panel** during installation. See “Default installation locations” on page 52 for information on where Tivoli Directory Integrator is usually installed.

Launching the appropriate installer

You can launch the IBM Tivoli Directory Integrator 7.1.1 Installer by using one of the following methods:

Launch the installer from the Launchpad

The Tivoli Directory Integrator Launchpad provides essential getting started installation information and links to more detailed information on various installation, migration, and post installation topics. In addition, Launchpad allows you to launch the Tivoli Directory Integrator installer.

Notes:

1. The Launchpad is not available on z/OS® and i5/OS.
2. Using the Launchpad requires that you have a supported Web browser installed and configured; if this is not the case, you cannot use the Launchpad. However, you can still use the platform-specific installer directly; see “Using the platform-specific TDI installer” on page 11 for instructions on how to use the Tivoli Directory Integrator Installer.

Note:

1. Open the Tivoli Directory Integrator Launchpad by typing the following command 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 windows. Click a menu item to view information about it. The following menu items are available:

Welcome

The installation Welcome window contains links to:

- IBM Tivoli Directory Integrator Web site
- 7.1.1 Documentation
- Support Web site
- Tivoli Directory Integrator news group



The following options on the left are Tivoli Directory Integrator Launchpad windows:

Release Information

This window 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 window contains links to information about platform support and hardware requirements.

Installation scenarios

This window contains a description of the TDI components available for installation. You can install some or all of these components during installation. This window also contains a description of the Password Synchronization Plug-ins components available for installation.

Migration Information

This window contains a link to information about migrating from Tivoli Directory Integrator 6.0, 6.1.X, 7.0, or 7.1 to 7.1.1. It also contains information about migrating the Derby System Store.

Install IBM Tivoli Directory Integrator

This window contains links to the IBM Tivoli Directory Integrator Installer, as well as links to installation, migration and supported platforms documentation. See “Using the platform-specific TDI installer” on page 11 for instructions on how to use the IBM Tivoli Directory Integrator Installer.

Install IBM Tivoli Directory Integrator Password Synchronization Plug-ins

This window 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 window is not available on Linux PPC and Linux 390 platforms.

Exit Exits the Launchpad, without installing anything.

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

Launch the installer directly

You can launch the installer directly using the installation executable file:

1. Locate the installation 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_tdiv711_win_x86.exe`

Windows 64-bit

`install_tdiv711_win_x86_64.exe`

AIX `install_tdiv711_aix_ppc.bin`

AIX 64-bit

`install_tdiv711_aix_ppc_64.bin`

Linux `install_tdiv711_linux_x86.bin`

Linux 64-bit

`install_tdiv711_linux_x86_64.bin`

Power PC Linux

`install_tdiv711_ppclinux.bin`

z/OS Linux

`install_tdiv711_zlinux.bin`

Solaris Sparc

`install_tdiv711_solaris_sparc.bin`

Solaris (Intel)

`install_tdiv711_solaris_x86_64.bin`

HP-UX Integrity

`install_tdiv711_hpxx_ia64.bin`

i5/OS INST_TDI.SH

2. Double-click the executable file, or type the executable file 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 “Using the platform-specific TDI installer.”

Note: Non-administrators can install Tivoli Directory Integrator, with the following caveats: users installing Tivoli Directory Integrator must have write privileges to the installation location; non-administrators do not see the “Register AMC as a service” and “Register Server as System Service” windows, and non-administrator Configuration Editor shortcuts differ from administrator Configuration Editor shortcuts.

Using the platform-specific TDI installer

The platform-specific Tivoli Directory Integrator installer is launched either from the Launchpad or from the command line. The Tivoli Directory Integrator installer can be used to install a new copy of Tivoli Directory Integrator, add a feature to an existing instance of Tivoli Directory Integrator, or upgrade a previous version of Tivoli Directory Integrator. The default install location on your computer for Tivoli Directory Integrator varies with the platform.

During installation, the Installer will log its actions in files residing in the system's temporary files directory, typically /tmp or /var/tmp on UNIX platforms.

Before you install

Note: In addition to being unavailable on the i5/OS operating system, Tivoli Directory Integrator supports neither the Configuration Editor (CE) nor the Configuration Editor Update Site on the following operating systems:

- HP-UX Integrity
- Solaris Opteron
- z/OS
- Linux PPC
- Linux 390

See "Using the Remote Configuration Editor" on page 131 for information on how to develop solutions without a local Configuration Editor.

Installing IBM Tivoli Directory Integrator on i5/OS

Tivoli Directory Integrator supports installation on i5/OS. The i5/OS platform does not support a GUI interface (Launchpad). i5/OS supports the command line installation -console option. See "Installing using the command line" on page 42.

The following Tivoli Directory Integrator features are not available on i5/OS, and are not listed as installable features during -console installation:

- Configuration Editor (CE/Integrated Development Environment (IDE)) – This component is the Tivoli Directory Integrator IDE. Also see "Using the Remote Configuration Editor on z/OS" on page 307.
- Configuration Editor Update Site (Eclipse Update site for CE) – An Eclipse update site used for CE maintenance and for allowing the customer to install the CE to an existing Eclipse workbench if they do not want to use the stand alone Rich Client Platform (RCP) application.

Prior to installing Tivoli Directory Integrator, certain software must already be installed on the i5/OS operating system running Tivoli Directory Integrator.

PTF versions: For i5/OS V6R1 the installer will check that the following items are installed:

1. Product 5761JV1 option 11 (J2SE 6.0 32 bit)
2. PTF group SF99562 level 19 or higher (Java)
3. Product 5761DG1, *BASE (IBM HTTP Server for i5/OS - contains LWI 8.1.0.3)
4. PTF group SF99115 is at level 18 or higher (LWI, AMC Role and SSL Fix)

For i5/OS V7R1 the installer will check that the following items are installed:

1. Product 5761JV1 option 11 is installed (J2SE 6.0 32 bit)
2. PTF group SF99572 is at level 08 or higher (Java)
3. Product 5770DG1, *BASE is installed (IBM HTTP Server for i5/OS - contains LWI 8.1.0.3)
4. PTF group SF99368 is at level 07 or higher (LWI, AMC Role and SSL Fix)

Java virtual machine version: Tivoli Directory Integrator requires the IBM J9 32-bit JVM on all versions of i5/OS. If the J9 32-bit JVM is not found or if not having minimum PTF group level, the following error message is displayed:

The install was unable to detect the IBM J9 VM (32-bit). The IBM J9 VM is required for this product. Please install this JVM then try again.

If you see this message, cancel the installation, install the IBM J9 VM, and begin the installation again.

If you choose the embedded Web platform feature, the install verifies to ensure that LWI 8.1.0.3 (v7r1 or v6r1) is resident on the target system.

If the PTFs or products are not found, the following error message is displayed:

The installer was unable to detect the i5/OS product/PTFs required by the embedded Web platform feature. You may choose to continue the install without the embedded Web platform feature or you may exit now and refer to the install log for a list of the missing requirements.

Installation:

Note: The installer and uninstaller on i5/OS are called INST_TDI.SH and uninstaller.sh, respectively.

To begin installing on i5/OS:

1. Locate the installation executable file for i5/OS in the TDI_INST directory on the product CD. Launchpad is not available on i5/OS. The default location to install i5/OS on your computer is: /QIBM/ProdData/IBM/TDI/V7.1.1
2. On i5/OS, in order to extract the Tivoli Directory Integrator 7.1.1 installer from a TAR image, you must set environment variable "QIBM_CCSID" to 819; that is, run the command
`export QIBM_CCSID=819`

before un-tarring the Tivoli Directory Integrator installer TAR image.

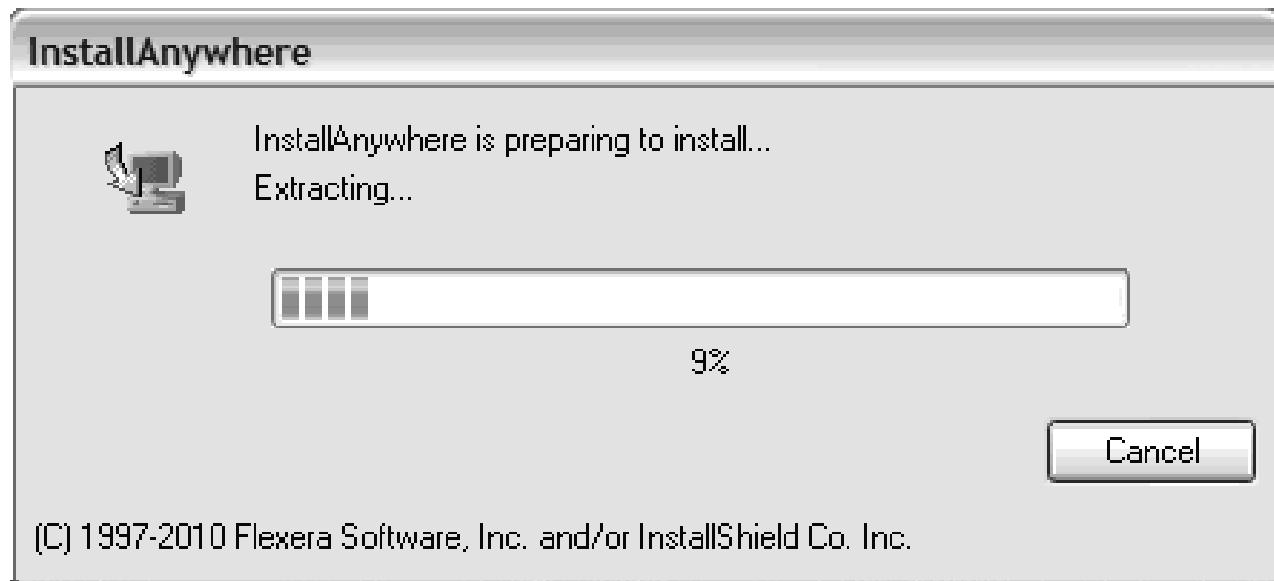
Another i5/OS difference is on the Tivoli Directory Integrator Solutions directory panel. On i5/OS, there is a specific place for user data. As a result, instead of giving you the option to make the installation directory the same as the solutions directory, the option reads: Use the TDI User Product Directory.

Installing using the graphical installer

Install Panel flow

Pre-Initialization Panel

You invoke the installer executable either from the command line, or by double clicking the executable (Windows only). This panel will initially appear followed by a splash screen:



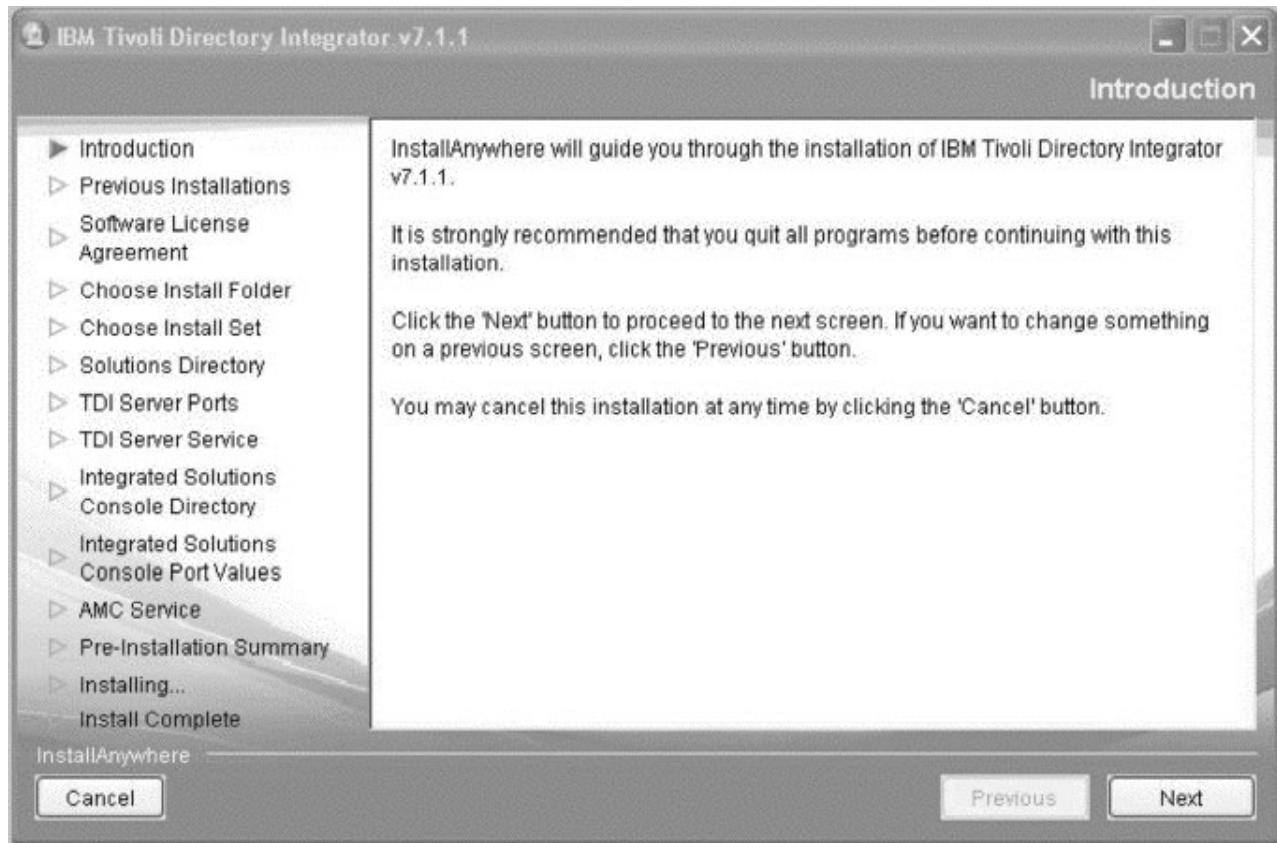
Note: The splashscreen may also show a drop down list of language choices if the underlying system supports more than one. (The default is English.)





The Welcome Panel

This is the Welcome Panel for the installer. This is the default panel provided by the InstallAnywhere installer. You have the option to continue by hitting the **Next** button or canceling out of the installer by pressing **Cancel**.



J9 PTF Missing Panel (i5/OS only)

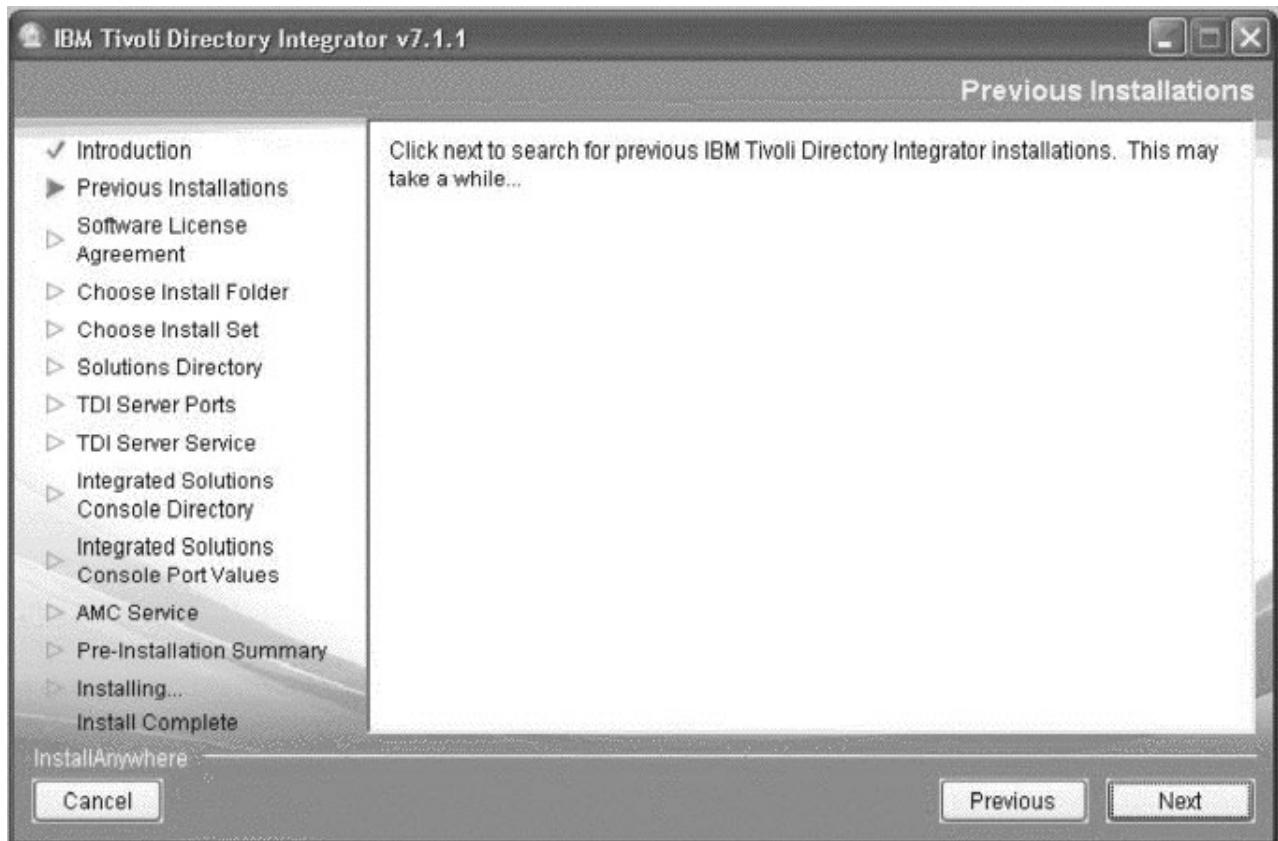
On i5/OS, the TDI install will check to make sure the IBM J9 32bit JVM is installed. If this JVM is not found, an error message will be displayed:

The install was unable to detect the IBM J9 VM (32-bit).
The IBM J9 VM is required for this product. Please install this JVM then try again.

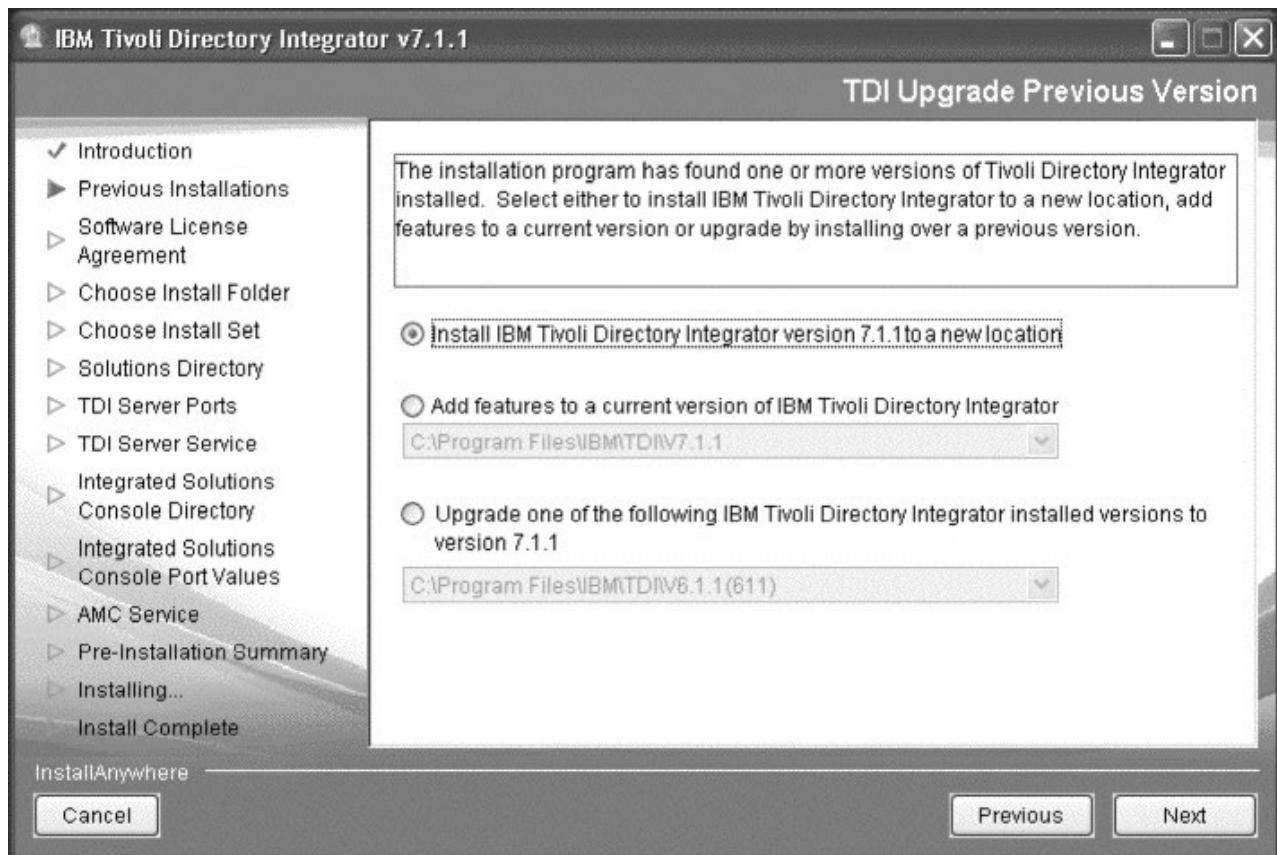
You will need to cancel the install at this point. This panel is not shown if the JRE check passes.

Previous Installed TDI Information Panel

This panel informs you that detecting previous versions of TDI may take some time.

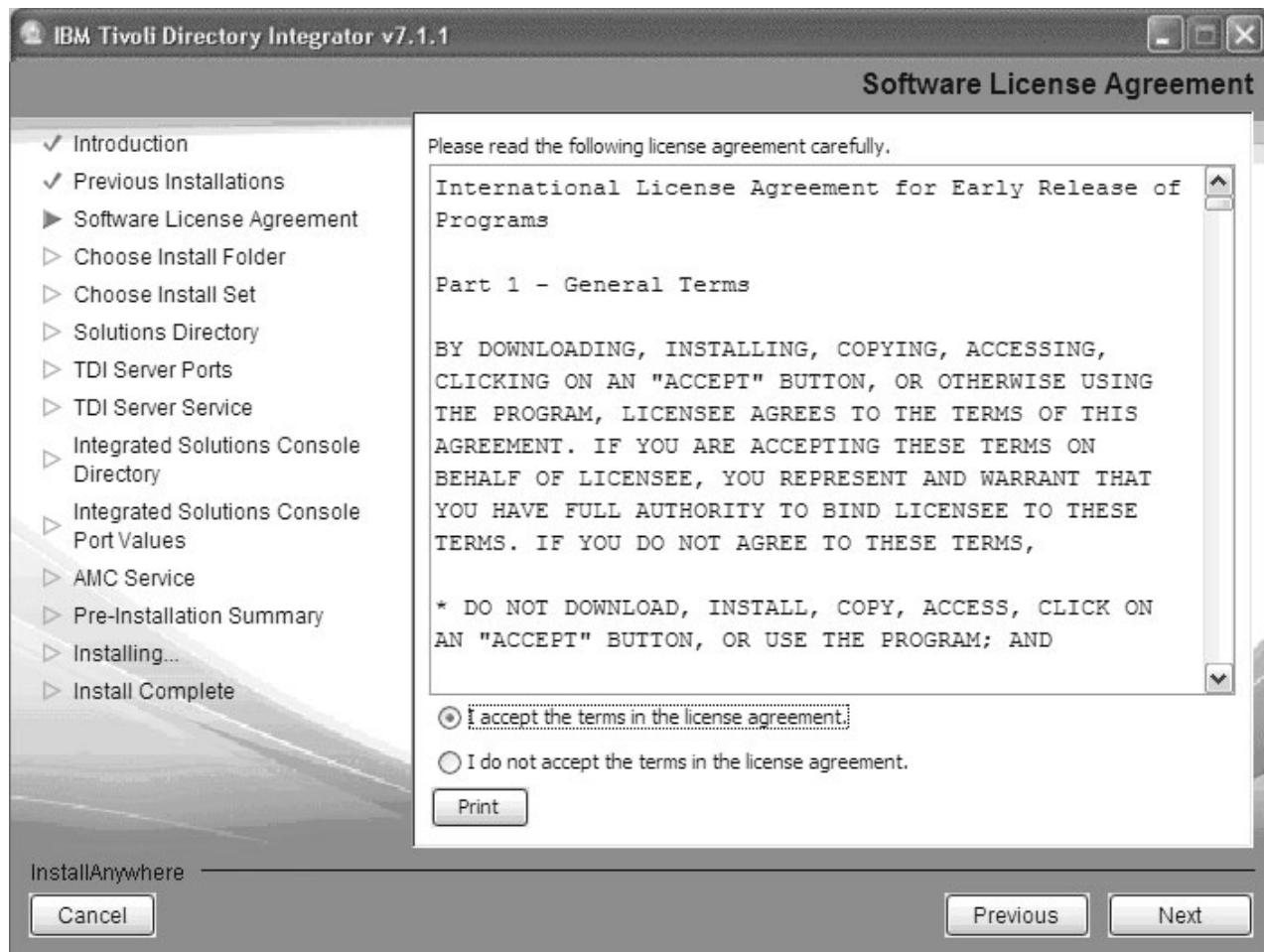


If a previous version is detected, you are presented with a number of upgrade options.



The License Panel

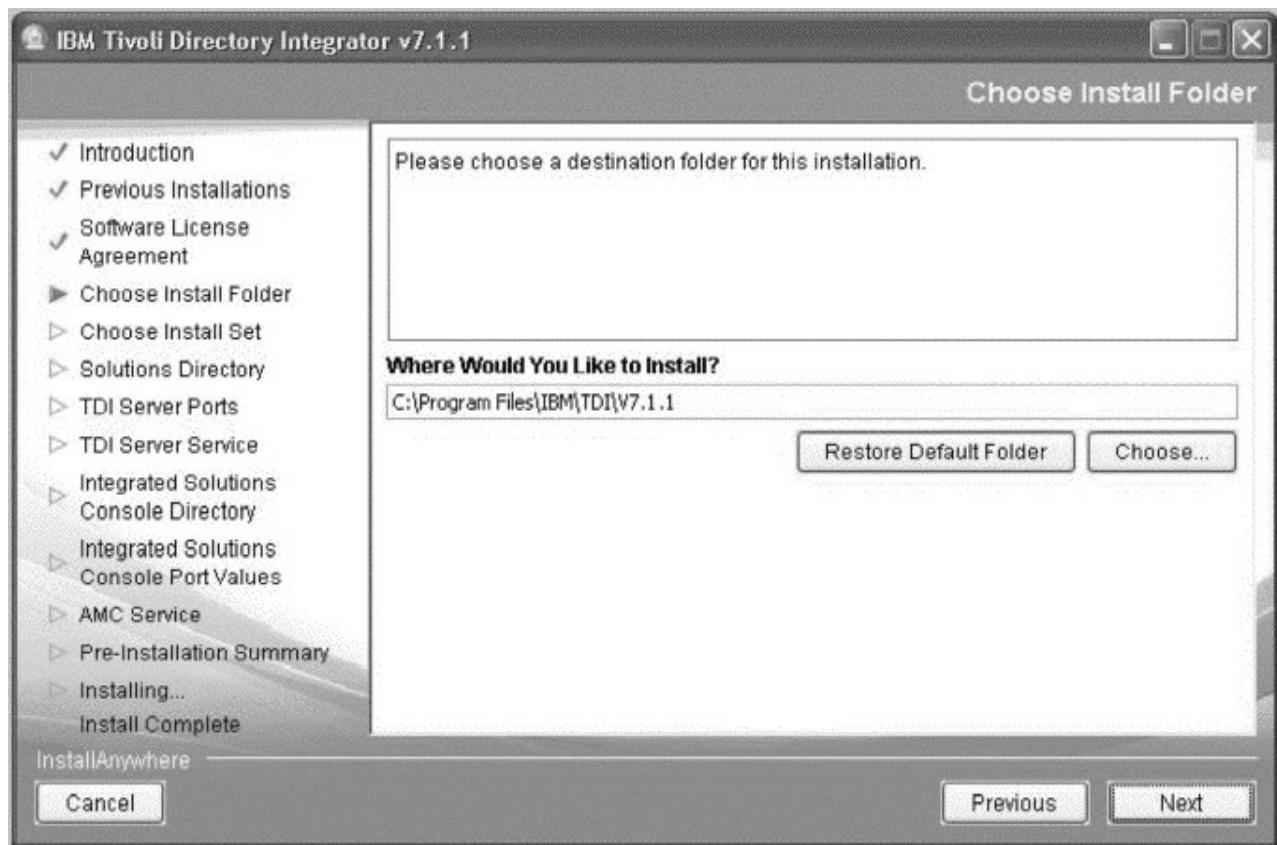
The License Panel is provided by the IBM license tool. This panel will be shown in a **New TDI v711 install** and **Upgrading an older TDI version**.



Destination Panel

Notes:

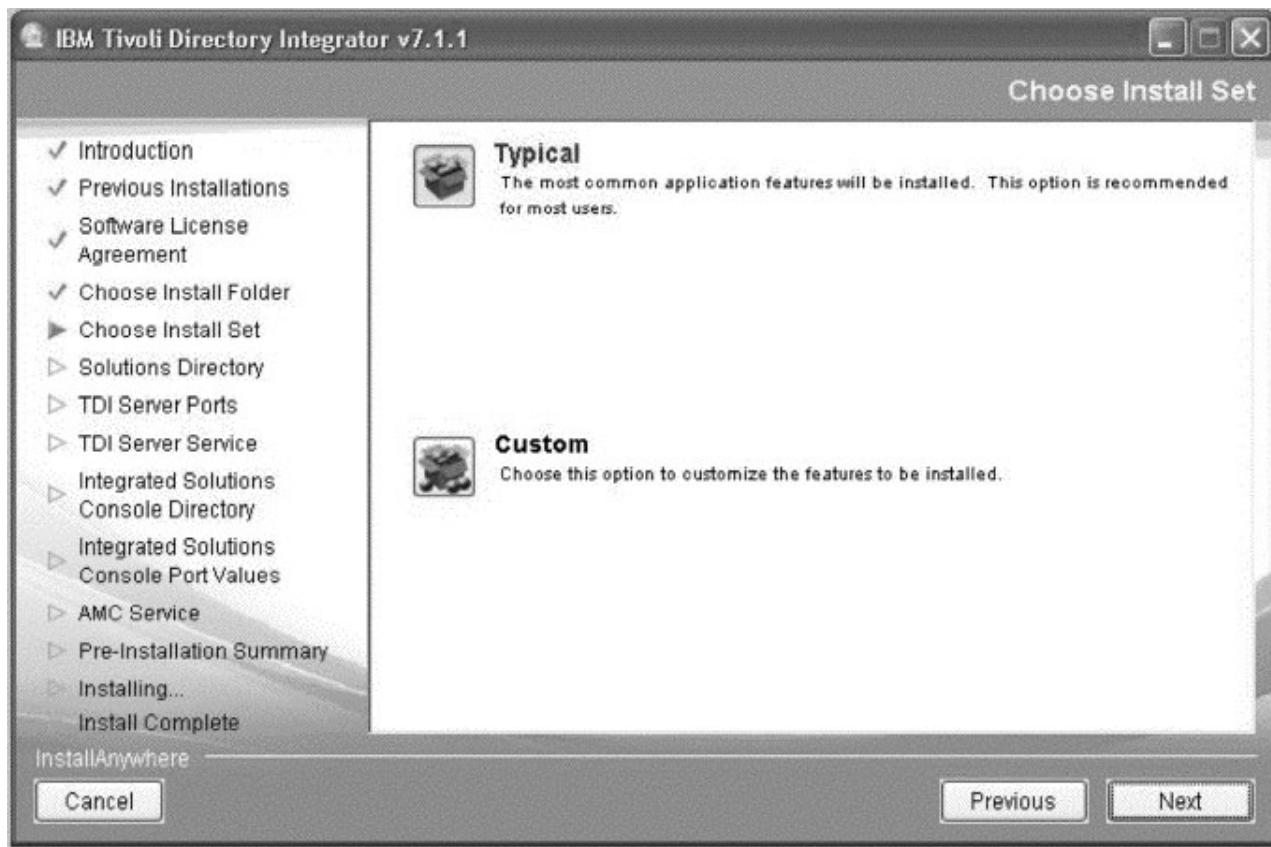
1. This panel will not be shown if an upgrade from TDI 6.0, TDI 6.1, TDI 6.1.1, TDI 7.0, or TDI 7.1 was selected nor will it be shown if you are adding features to an existing TDI 7.1.1 instance.
2. The destination panel will have the last value entered if you go forward in the wizard to other panels and then come back.
3. Non-ASCII characters and the following list of characters are not supported in the install path:
";|*?!#&\$',=^@%+



Install Type Panel

The Typical install includes the Runtime Server, the Configuration Editor (CE), Javadocs, Examples and AMC. It does not include the Configuration Editor Update Site, IBM User Interface Help System built on Eclipse, or the Password Synchronization Plug-ins.

If you select **Typical**, the feature selection panel is skipped. Also, you will automatically get the bundled embedded Web platform/ISC package. The ISC Directory panel will be skipped.

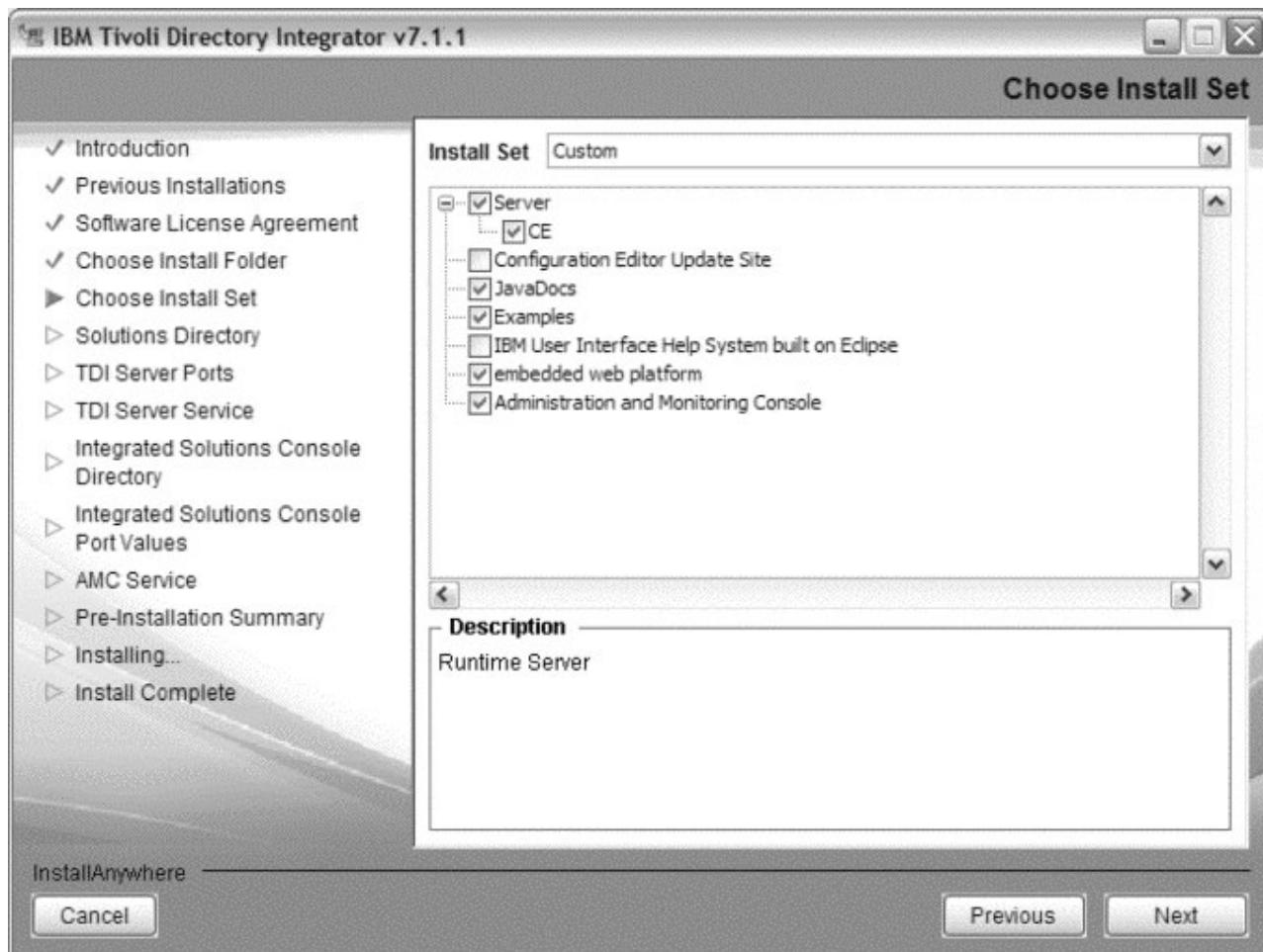


Feature Selection Panel

This panel lets you specify which features will be installed. Any feature can be individually installed if needed. The only exception to this is that if the Configuration Editor is selected, the server will be selected because the Configuration Editor is a subfeature of the server.

If any feature is not supported on the platform it will not be shown on the feature selection panel.

Note: The Configuration Editor feature is not available on zLinux, Linux PPC, Solaris Opteron, HP IA64 or i5/OS. The Configuration Editor Update Site feature is not available on zLinux, Linux PPC, Solaris Opteron, or i5/OS. The Password Synchronization Plug-ins feature is not available in the General Purpose edition nor on zLinux, Linux PPC, Solaris Opteron, HP-UX IA64 or i5/OS.



The following list summarizes each feature:

Runtime Server

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

Configuration Editor

A development environment for creating, debugging and enhancing TDI integration solutions. (Not available on zLinux, Linux PPC, Solaris Opteron, HP-UX or i5/OS.) This feature can not be installed without installing the Runtime Server.

Configuration Editor Update Site

Patterned after the Eclipse Update Site. Contains the necessary files to install the Config Editor to an existing Eclipse. It will also be used for maintenance. (Not available on zLinux, Linux PPC, Solaris Opteron or i5/OS.)

Javadocs

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

Examples

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

IBM User Interface Help System built on Eclipse, v3.3.1 (local help)

An IBM User Interface Help System (previously known as IEHS) built on Eclipse that you

can install locally as an alternative to using the global online help service. This option requires manual download and deployment of Tivoli Directory Integrator help files after installation.

embedded web platform

The embedded Web platform 8.1 package; this version includes ISC SE.

Administration and Monitoring Console

A browser-based application for monitoring and managing running Tivoli Directory Integrator Servers.

Password Synchronization Plug-ins

TDI password synchronization plug-ins. (Not available in the General Purpose edition nor on zLinux, Linux PPC, Solaris Opteron, HP-UX or i5/OS).

Missing embedded web platform pre-req Panel (i5/OS only)

If the installer is missing the embedded web platform pre-requisites on i5OS and the embedded web platform feature is chosen (either by selecting it on the custom feature panel or by choosing a typical install), it will display the message:

The installer was unable to detect the i5/OS product/PTFs required by the embedded web platform feature. You may choose to continue the install without the embedded web platform feature or you may exit now and refer to the install log for a list of missing requirements.

You can choose to go back and deselect the embedded web platform feature through the custom feature panel and continue, or to exit the installer and get the appropriate products installed.

TDI Solutions Directory Panel

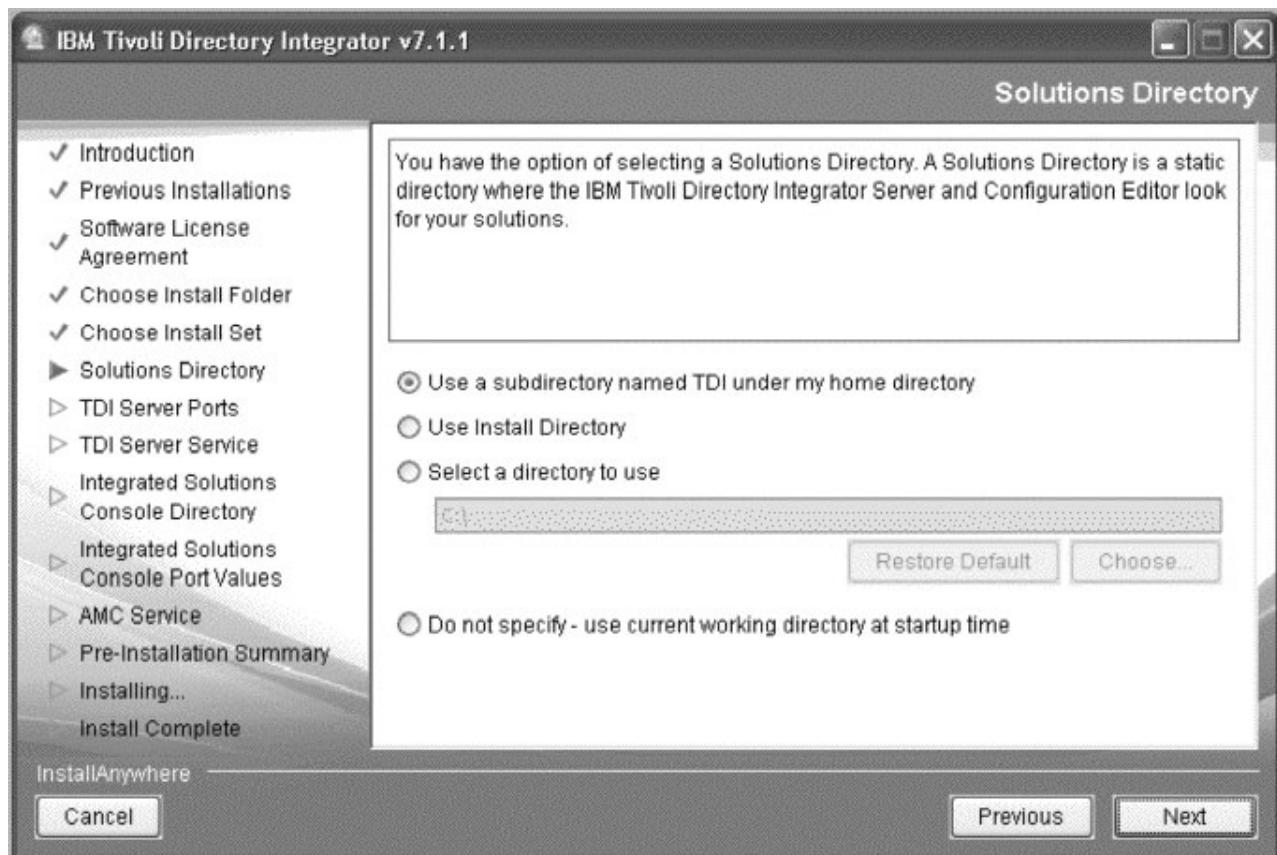
This panel is only displayed if the Server feature was chosen. It lets the user decide where the default Solution Directory the Server and Configuration Editor will search under. The Solution Directory is a static directory containing the solutions created by the user that will be run. By default, this panel will select to have the Solution Directory set the user's home directory.

From TDI 7.1.1 onwards, for Windows and UNIX platforms, if you select the **Select a directory to use** radio button, you need to specify a valid Solution Directory. The Universal Naming Convention (UNC) path is supported for Solution Directory during installation time.

On i5/OS (console only), the **Use Install Directory** will be replaced with **Use the TDI User Product Directory**. The i5/OS platform has a specific location (/QIBM/UserData) for user data. The install directory is not appropriate.

Note: This panel will not be shown in an upgrade from TDI 6.0, TDI 6.1, TDI 6.1.1, TDI 7.0, or TDI 7.1.

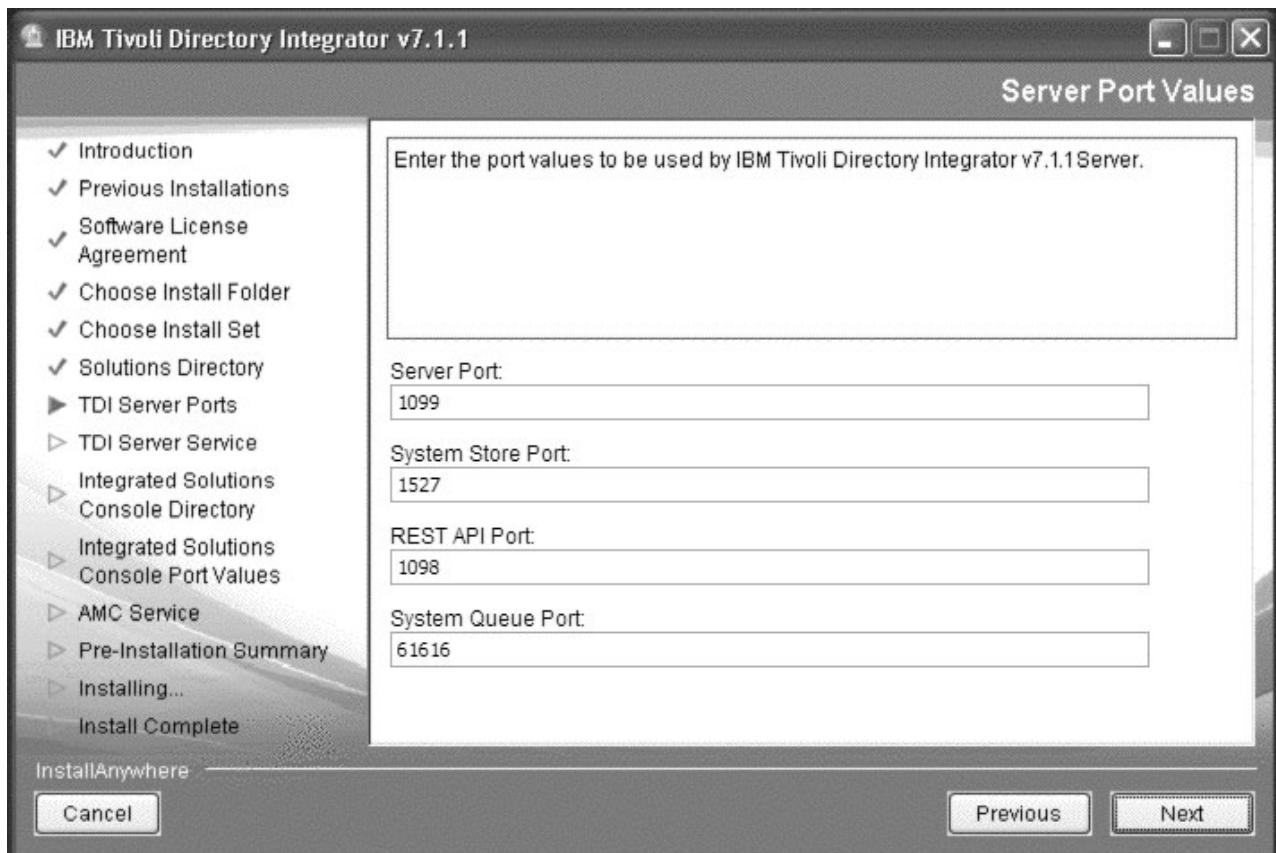
If you are adding features, and the Server feature was already installed, this panel will not be shown.



Server Port Configuration Panel

This panel will only be displayed if new instance of TDI 7.1.1 is getting installed and you have selected the Server to install as a feature or if it is an upgrade installation.

You will be asked for 4 various server ports numbers. There will be default values for these ports. The installer will make sure that you enter a valid and available port number (see Server Port Configuration).



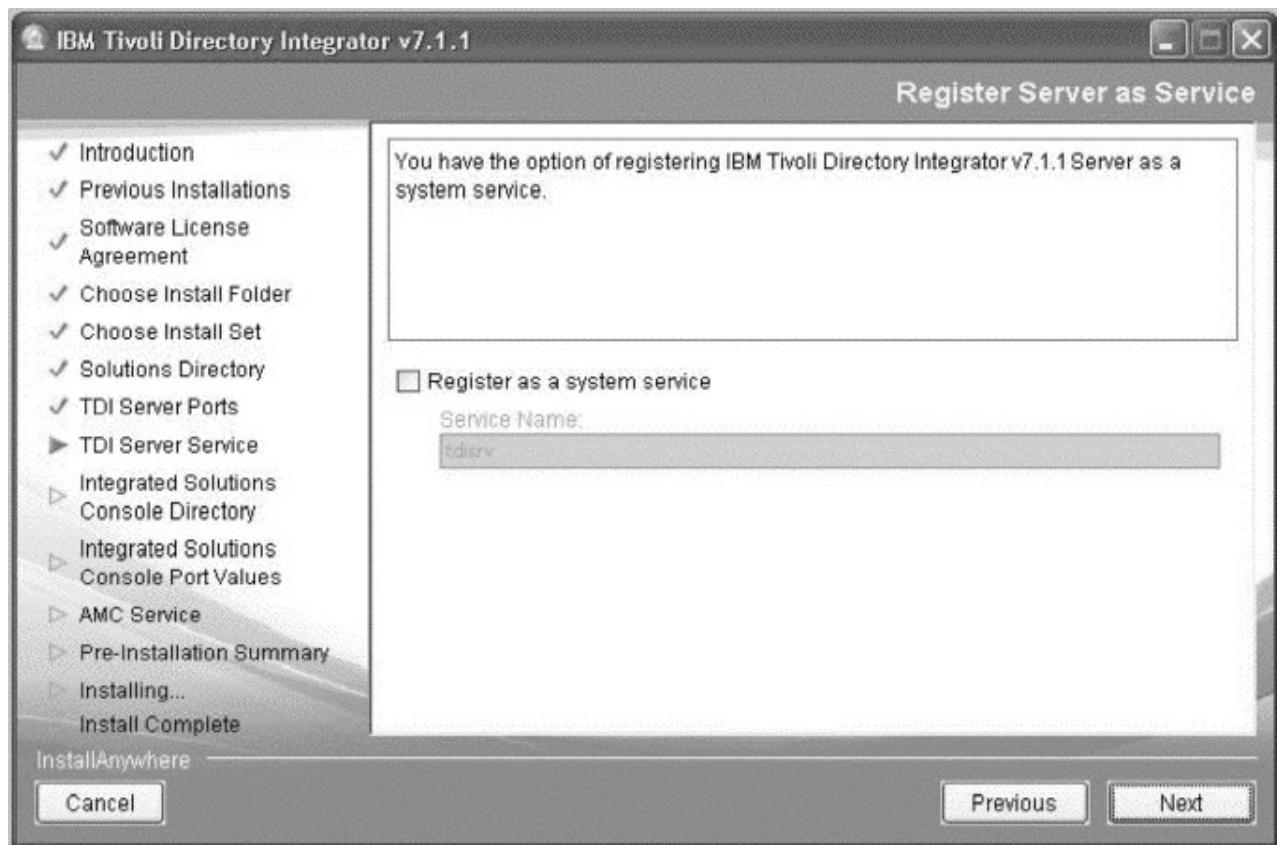
Register Server as a System service panel

This panel will only be displayed if new instance of TDI 7.1.1 is getting installed and you have selected the Server to install as a feature or if it is an upgrade installation. Also this panel will only be displayed if you have Administrative privileges.

If the checkbox is checked, then only SERVER will be registered as a service for that OS.

The default is for the checkbox to be unchecked. The two text boxes will be enabled only if the checkbox is checked. The first text box is for service name and the second is for the port number that the server as a system service will use to run on.

The installer will do its best to provide a valid default value for Service Name (see Registering Server as a Windows service or Unix Process for details on this process). If the installer is unable to determine a valid Service Name, the field will be blank. You will not be able to move forward until you enter a valid service name.



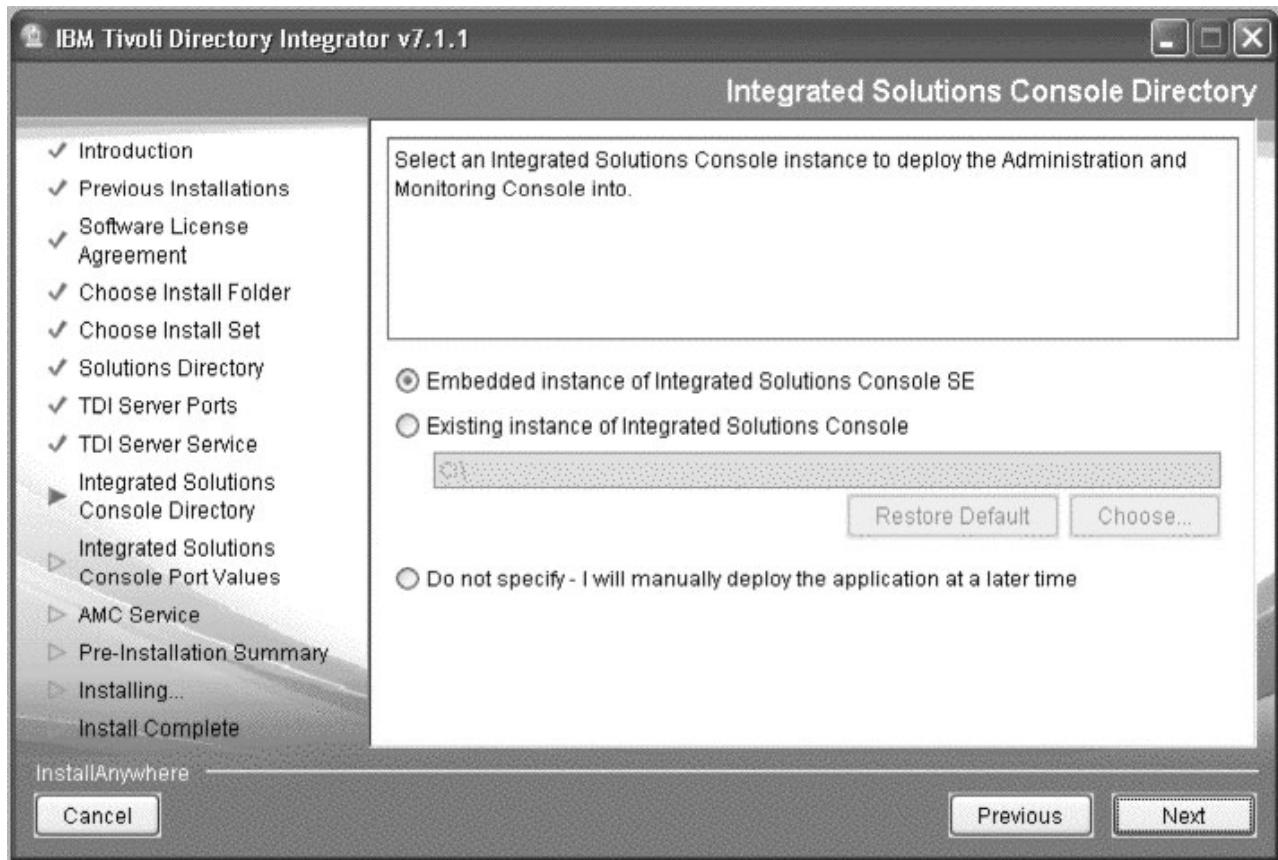
TDI AMC Deployment Panel

This panel is only displayed if the Custom install set was chosen and you also chose to install the AMC feature. You must choose which ISC instance AMC will be deployed to. You may choose to deploy AMC to the bundled ISC that is shipped with TDI, an ISC that is already installed on the target machine, or choose to deploy AMC at a later time. When choosing an ISC that is already installed, the user must select a directory that contains the embedded Web platform (LWI) or WAS, for example C:\Program Files\IBM\WebSphere\AppServer or C:\dev\IBM\TDI\lwi.

If you did not choose to install the embedded web platform feature, then that choice will be grayed out.

Notes:

1. If you are adding features and the AMC feature is already installed, this panel will be skipped.
2. When deploying AMC to WAS the TDI AMC Admin role is not assigned automatically as when deploying to the embedded Web platform. This role must be manually assigned by the ISC console administrator.



ISC Port panel

This panel is shown either during a typical install or custom install, when you choose to deploy AMC to an Embedded instance of ISC. The ISC instance could be the embedded ISC that is shipped with Tivoli Directory Integrator 7.1.1, or it can be an ISC that is already resident on the target system.

If you are deploying AMC to a custom SE, the default values that are used for the HTTP and HTTPS ports are found as follows:

Look in the *TDI_Selected_ISC/conf/overrides/*.properties* files for the first occurrence of the properties *com.ibm.pvc.webcontainer.port* and *com.ibm.pvc.webcontainer.port.secure* and use the associated values. If either of these properties is not defined in any of the .properties files in that directory, look in *TDI_Selected_ISC/conf/config.properties* for them. If the HTTP port is not found, it will default to port 80, and if the HTTPS port is not found, it will default to port 443. The help port will have the same value as the HTTP port.

If you are deploying AMC to a custom AE, the default values that are used for the HTTP and HTTPS ports are found as follows (except on i5/OS where the defaults are taken):

Look for files named *serverindex.xml* file in the following directory specification:

TDI_Selected_ISC\profiles\AppSrv01\config\cells\nodes**. Inside those files, look for XML blocks similar to the following for the HTTP port:

```
<specialEndpoints xmi:id="NamedEndPoint_1200476459036" endPointName="WC_adminhost">
  <endPoint xmi:id="EndPoint_1200476459036" host="*" port="9060"/>
</specialEndpoints>
```

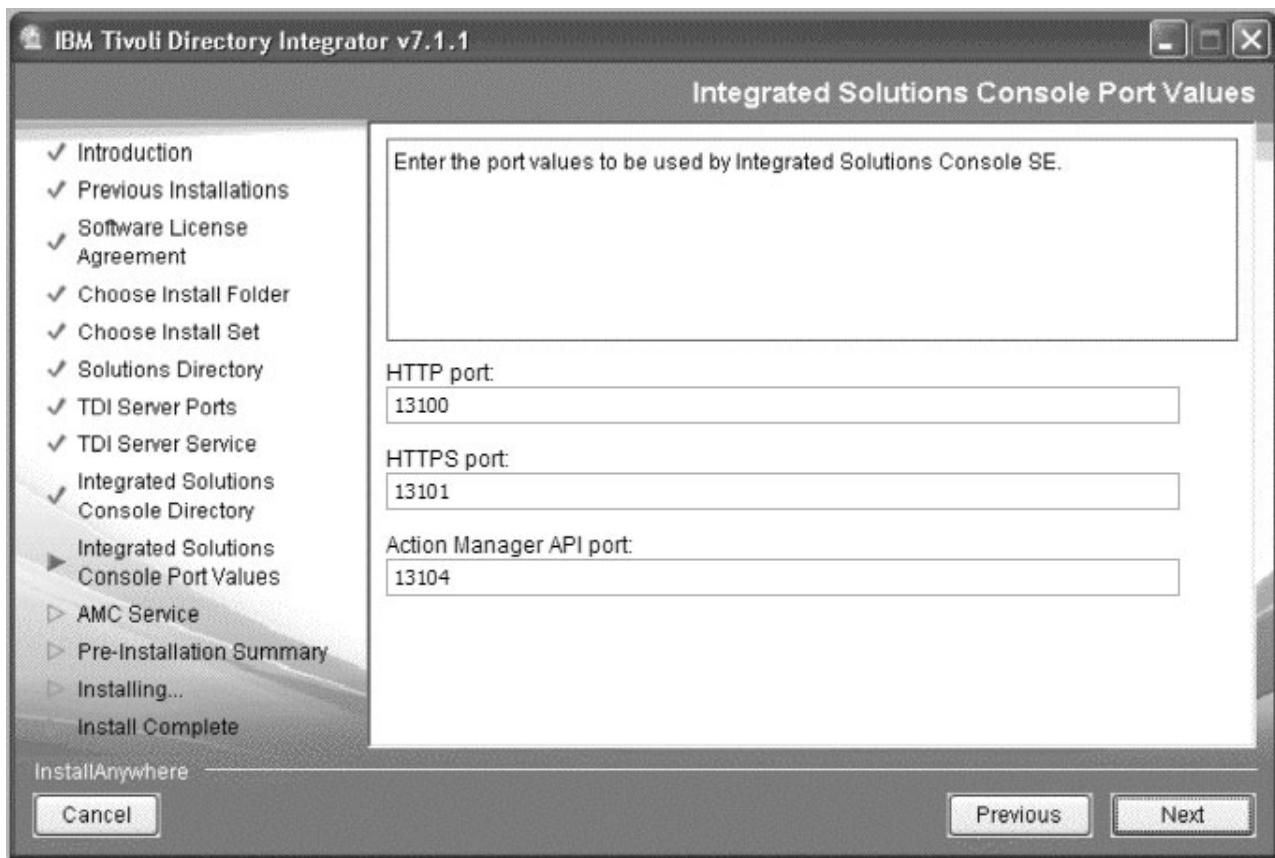
and similar to the following for the HTTPS port:

```
<specialEndpoints xmi:id="NamedEndPoint_1200476459039" endPointName="WC_adminhost_secure">
  <endPoint xmi:id="EndPoint_1200476459039" host="*" port="9043"/>
</specialEndpoints>
```

The installer searches for a specialEndpoints tag that has an endPointName of **WC_adminhost** or **WC_adminhost_secure** and use the associated port values from the embedded endPoint tags. In the event the HTTP port is not found by this method, it 9060 and in the event the HTTPS port is not found, it will default to 9043. The help port will be set to the HTTP port value.

The values shown are the defaults for the embedded SE.

The panel will not allow ports to be entered that are already in use. A warning message will appear asking you to choose another port value.

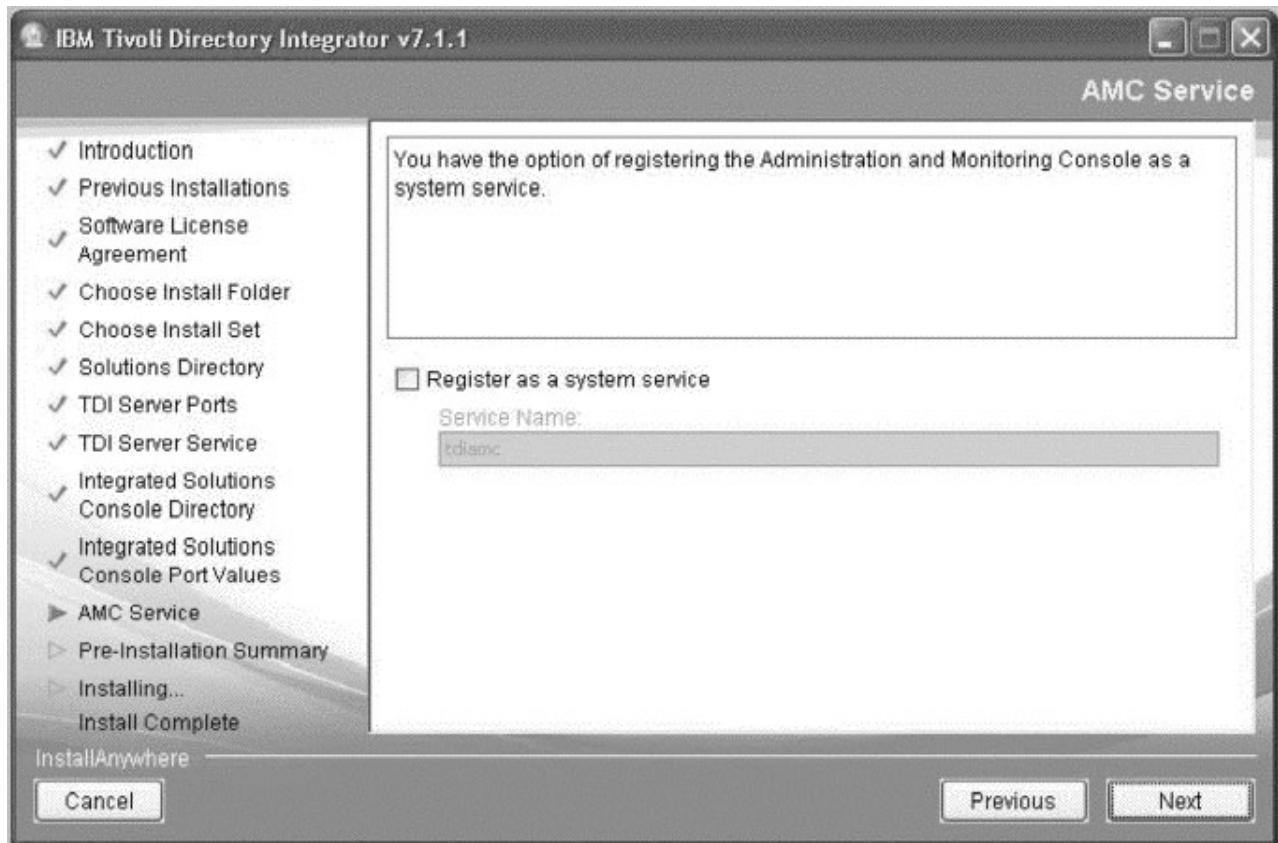


Register AMC as a service Panel

If the checkbox is checked, then AMC will be registered as a service for that OS.

The default is for the checkbox to be unchecked.

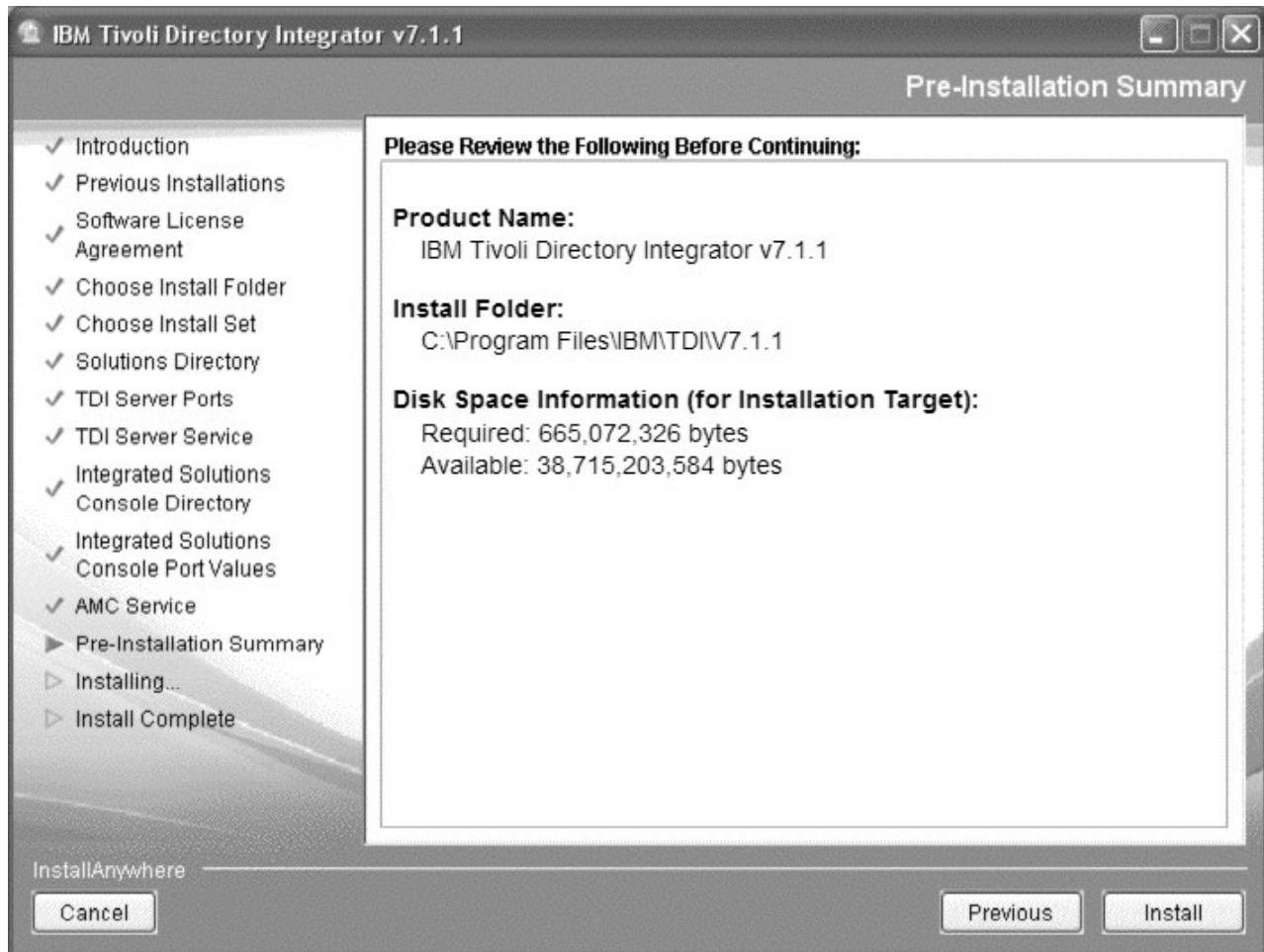
This panel is only shown if the embedded web platform and AMC features were selected and if you have administrative privileges.



The installer will do its best to provide a valid default value for Service Name (see Registering AMC as a Windows service or Unix process for details on this process). If the installer is unable to determine a valid Service Name, the field will be blank. You cannot move forward until you enter a valid service name.

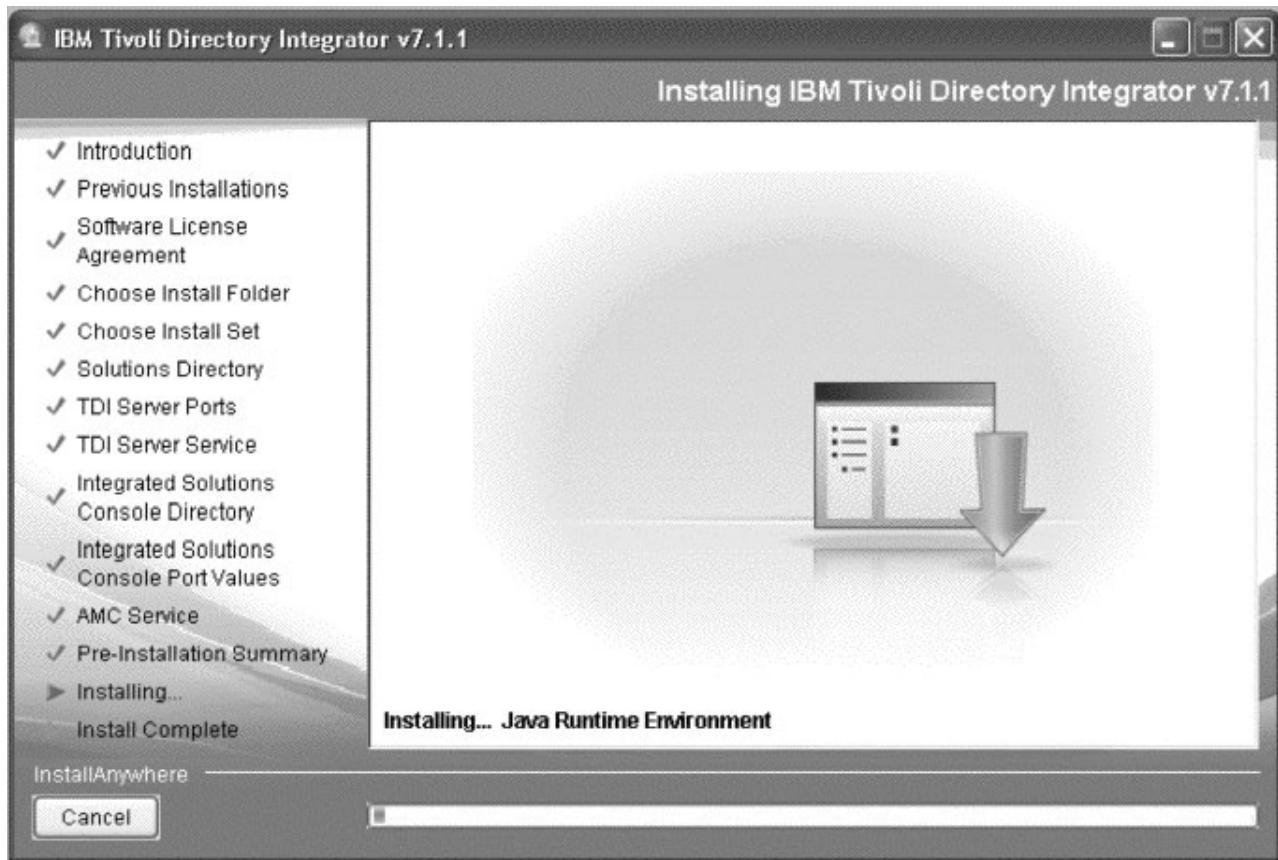
Pre-Install Summary Panel

This Summary panel gives you a summary of what features will be installed and where they will be installed to.



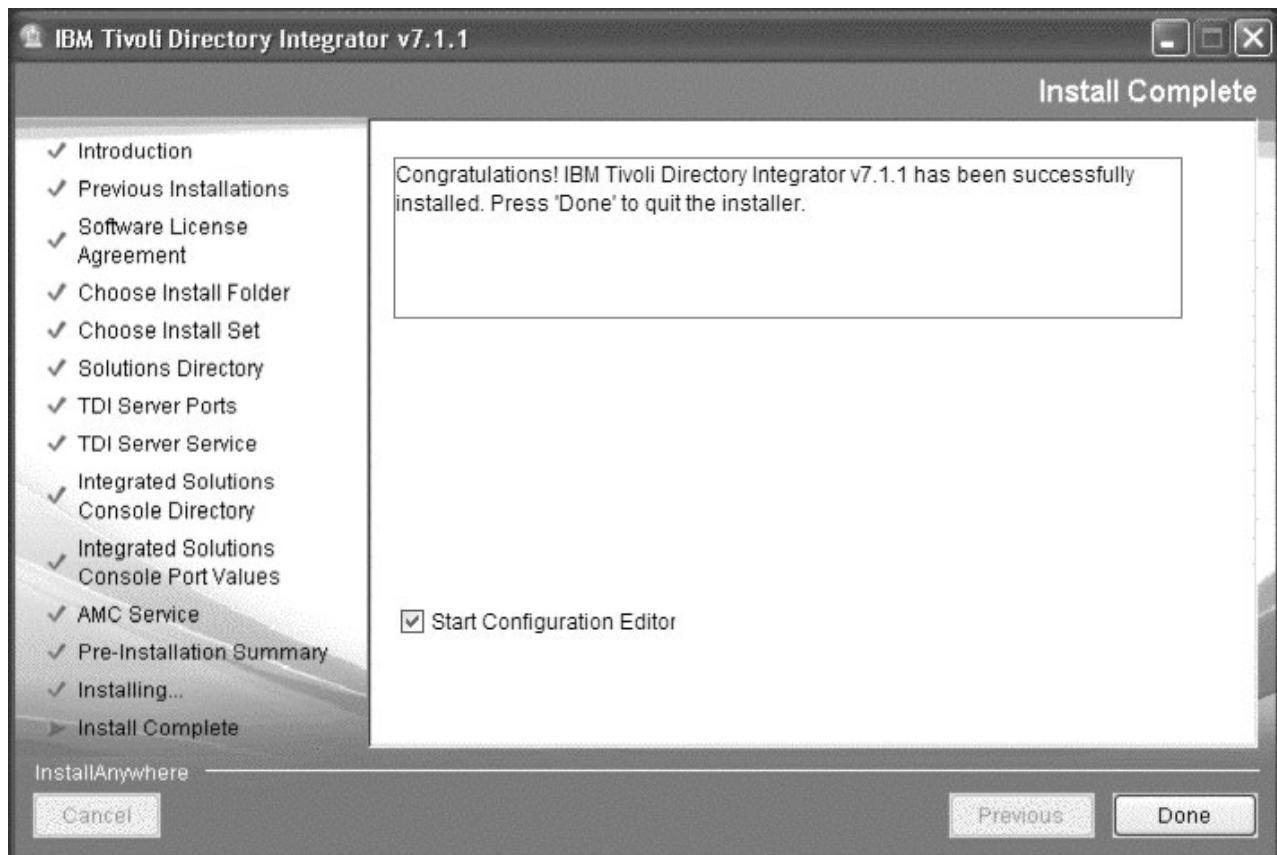
Installation Progress Panel

This panel is displayed while the actual install is occurring. This panel is the Progress Panel provided by InstallAnywhere. All of the features are installed while this is occurring.



Installation Complete Panel

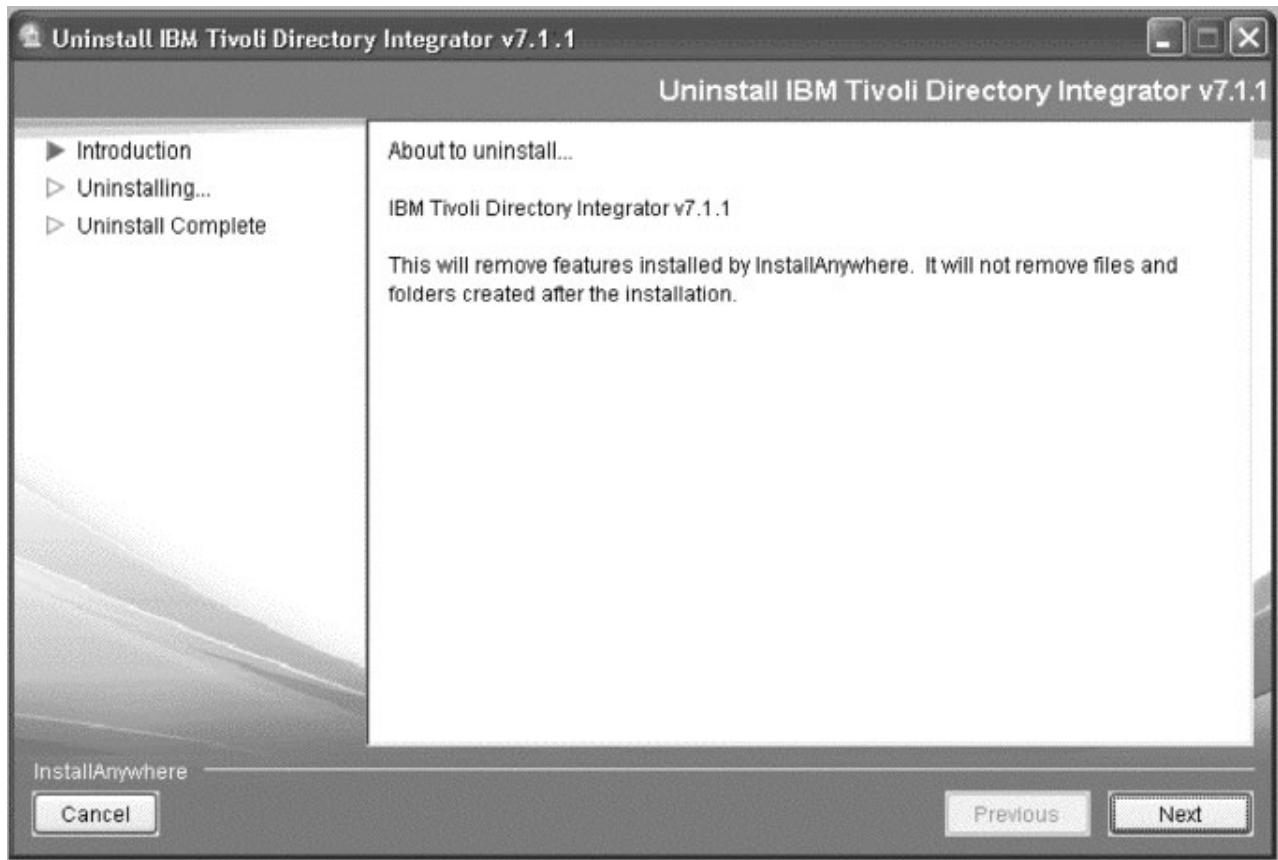
This panel shows you that the install has completed successfully. When the Done button is pressed, the install is complete. **Start the Configuration Editor** is checked by default.



Uninstall Panel flow

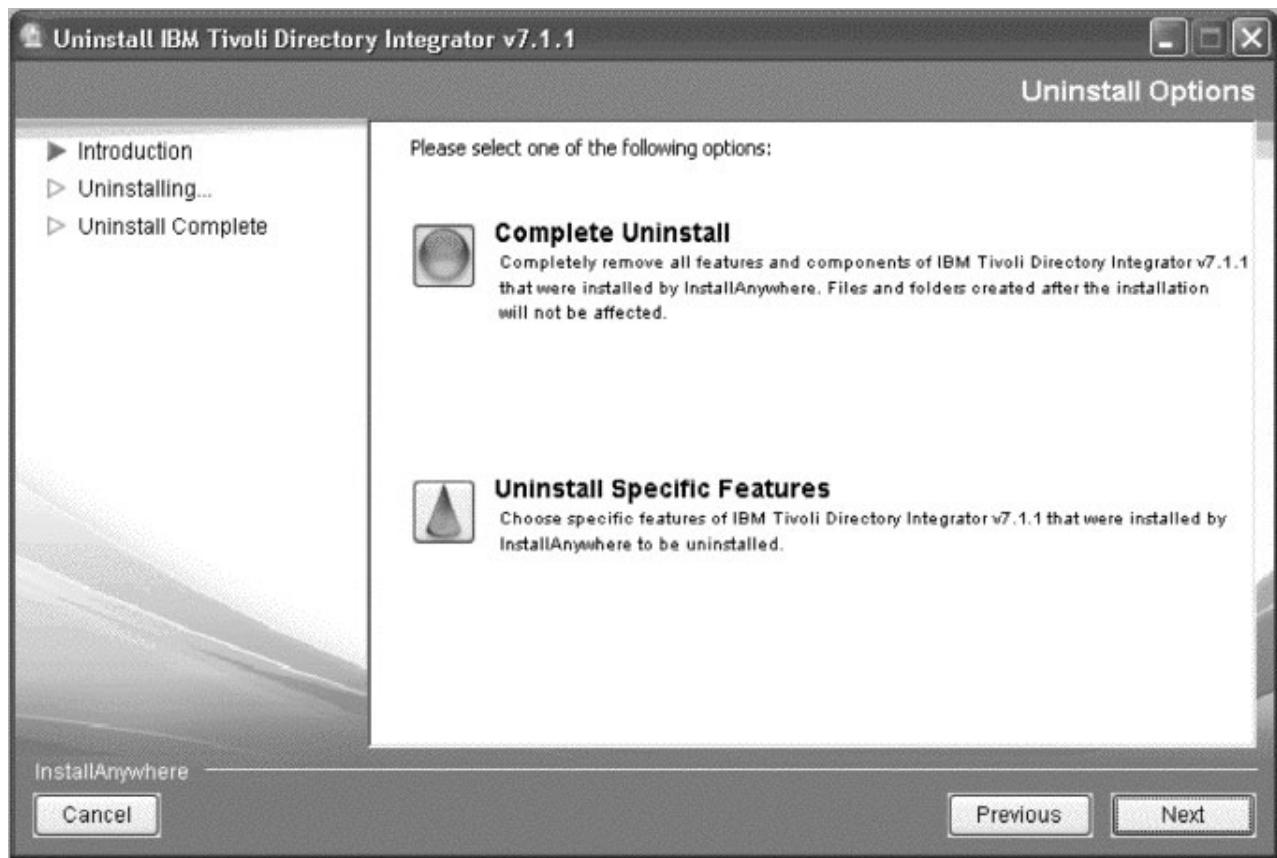
Uninstall Welcome Panel

This is an InstallAnywhere panel, with standard content.

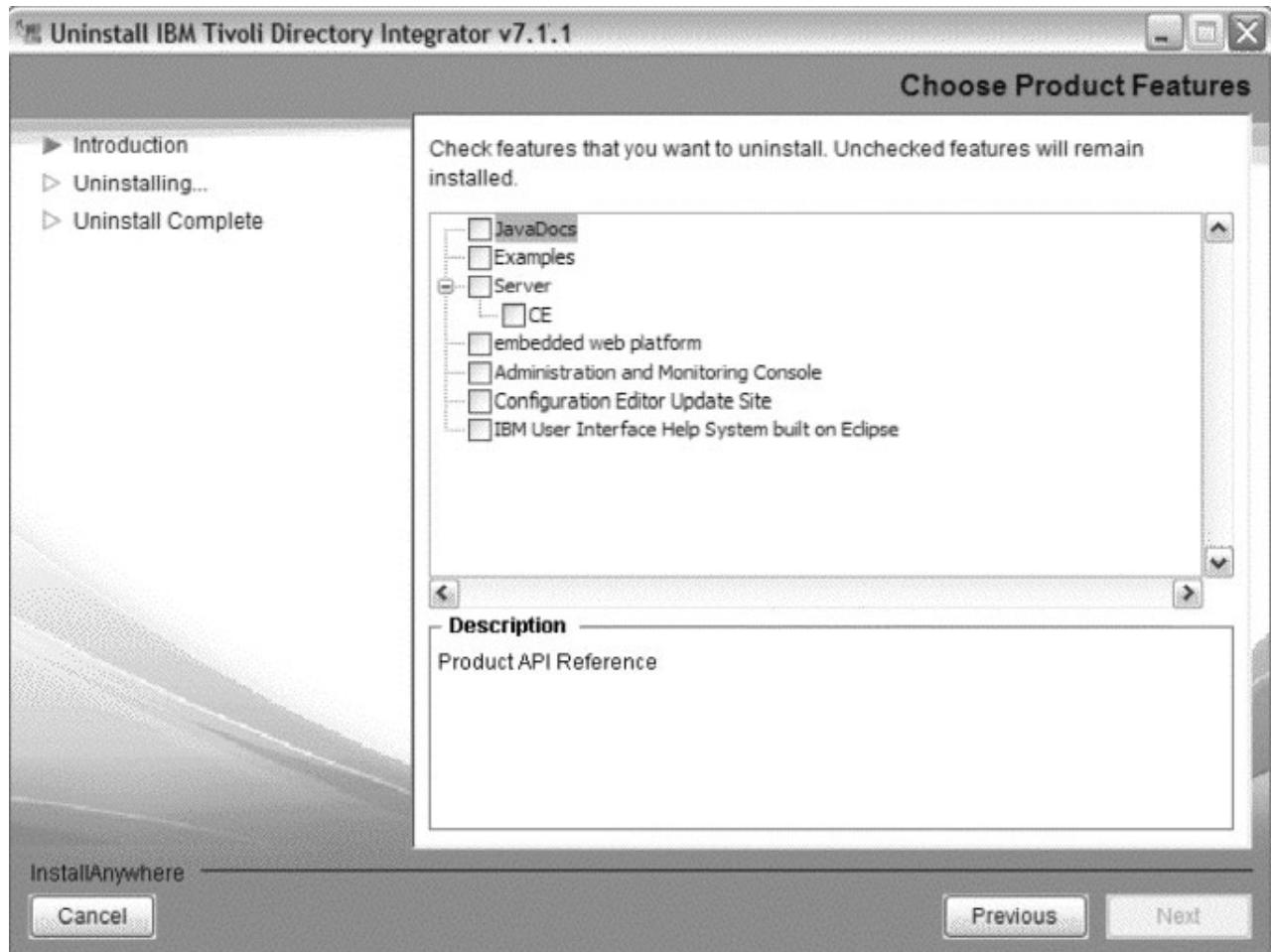


Choose Product Features Panel

This panel allows you to choose to uninstall the entire product, or only specific features.

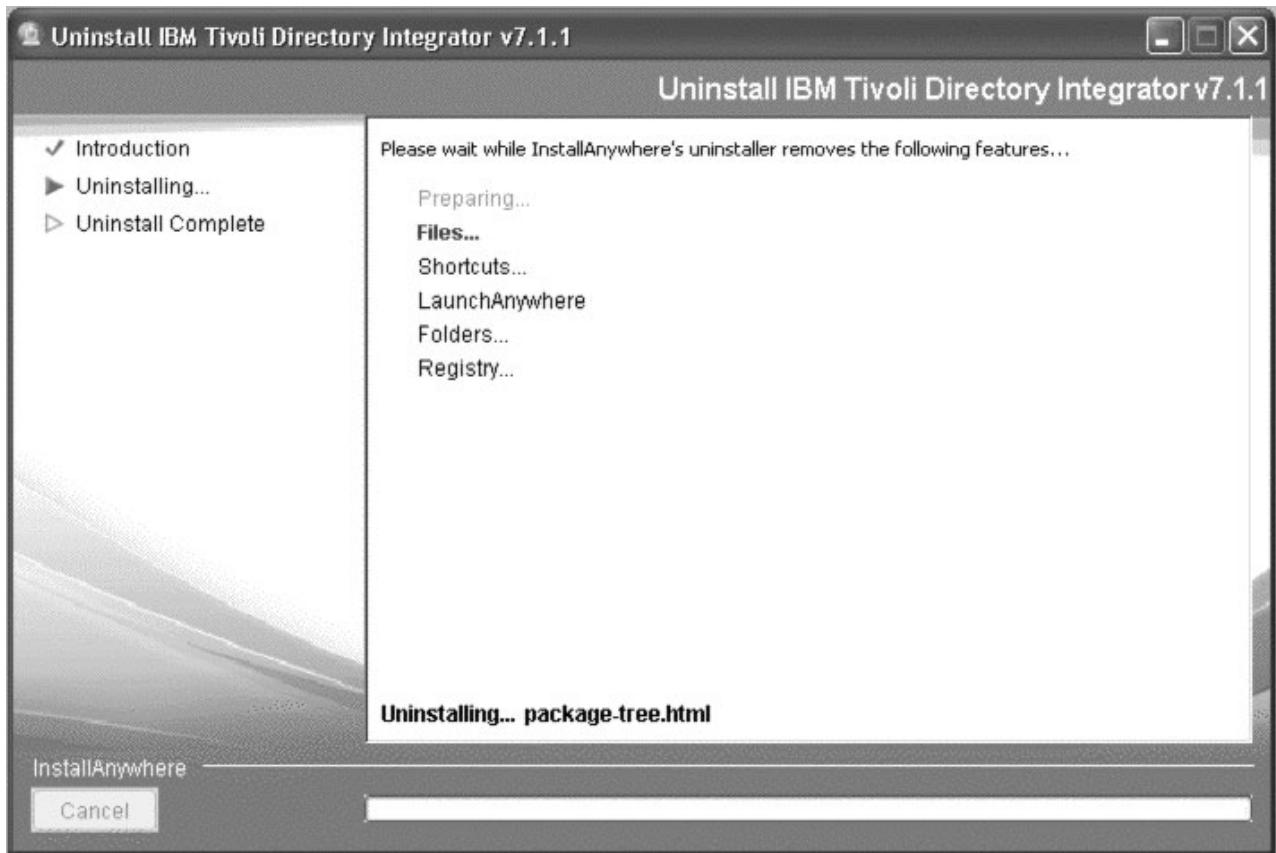


If **Uninstall Specific Features** is chosen, the following panel is also displayed:



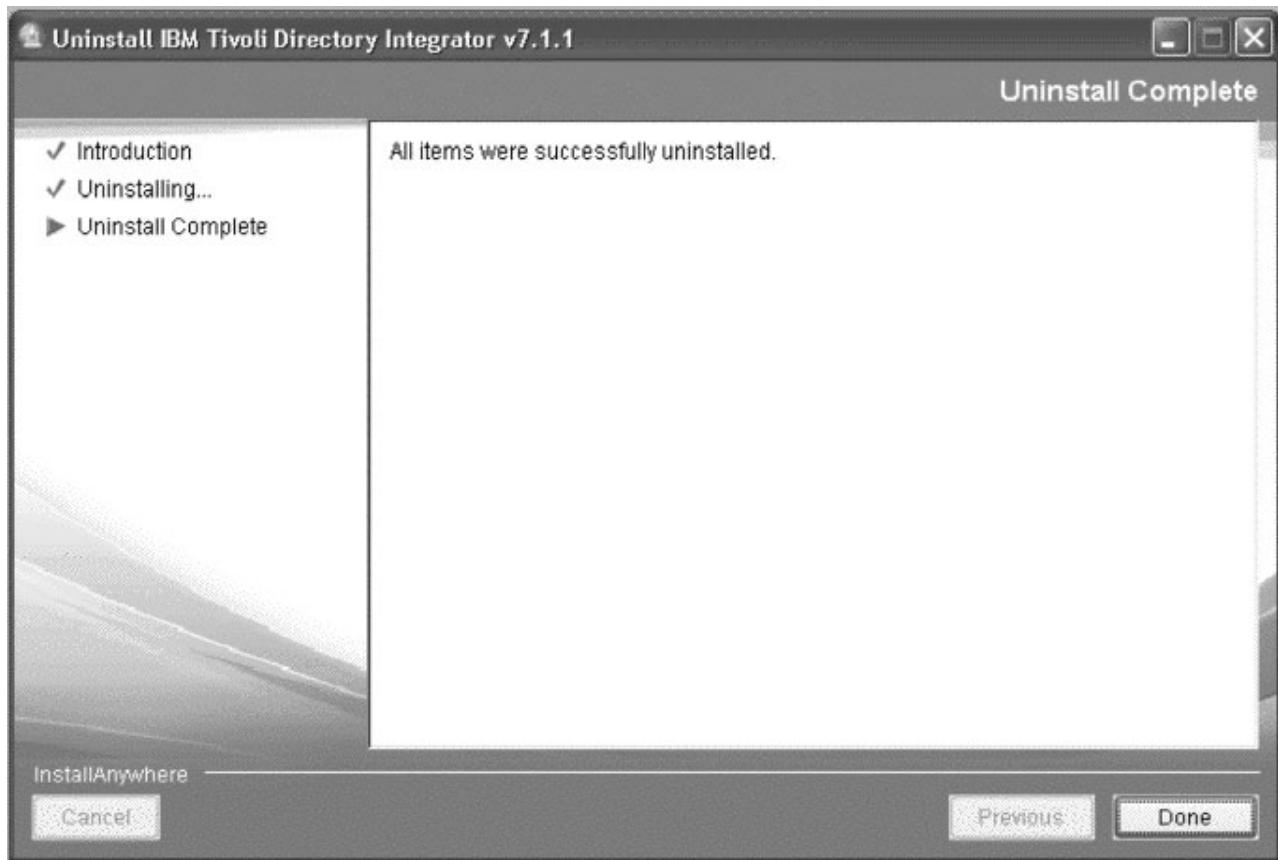
Uninstall Progress Panel

This panel is shown during uninstallation.



Uninstall Finish Panel

This panel shows you that the uninstallation has completed successfully. When the Done button is pressed, the uninstaller exits.



Add Feature Panel flow

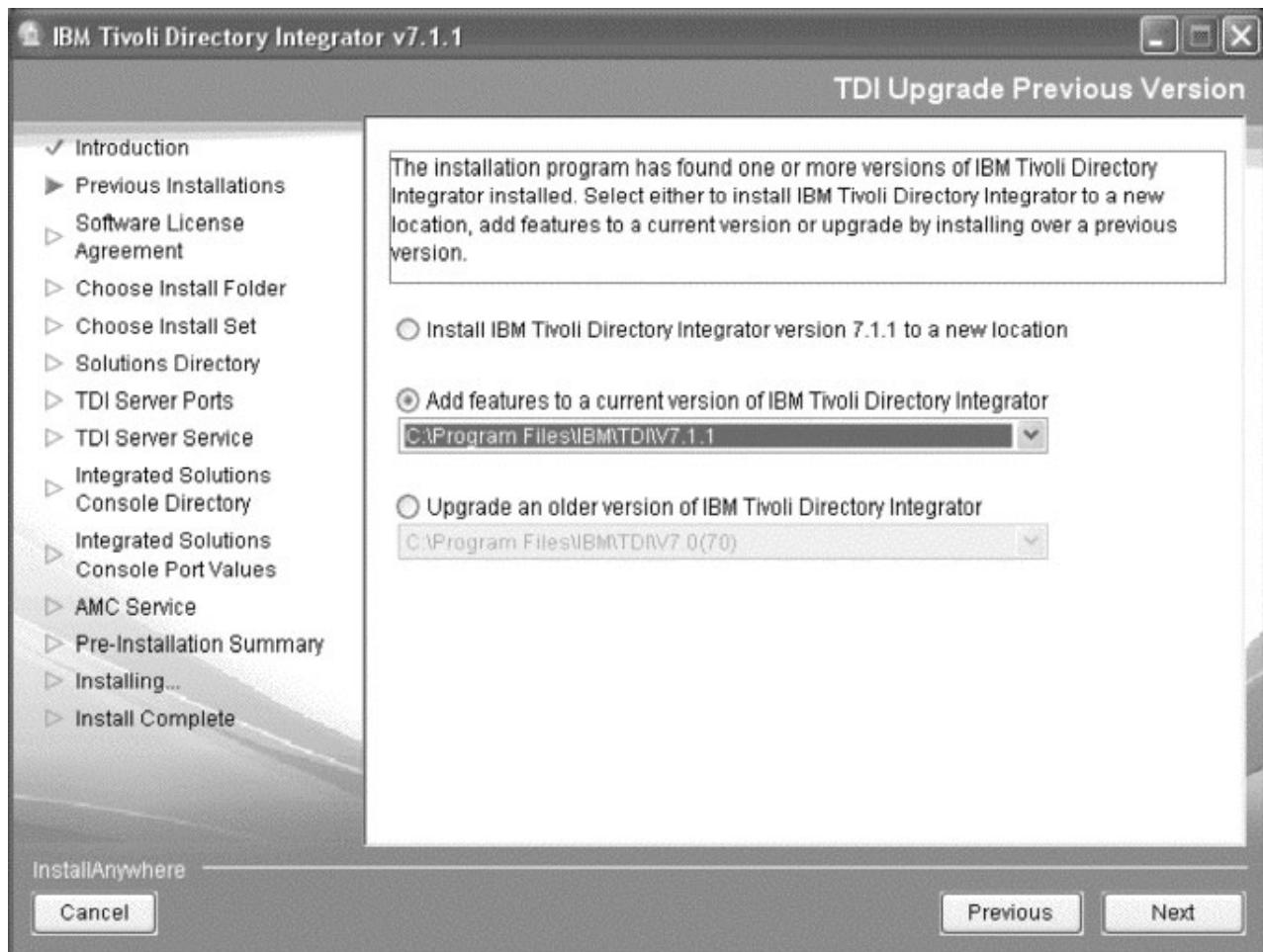
The Add Feature flow is similar to the new install flow. Only the unique panels will be shown here.

Pre-Initialization Panel

The Welcome Panel

Upgrade Panel

After the Welcome panel and the Previous TDI information panel, if there is an instance of TDI already installed on the box, you will see this panel.



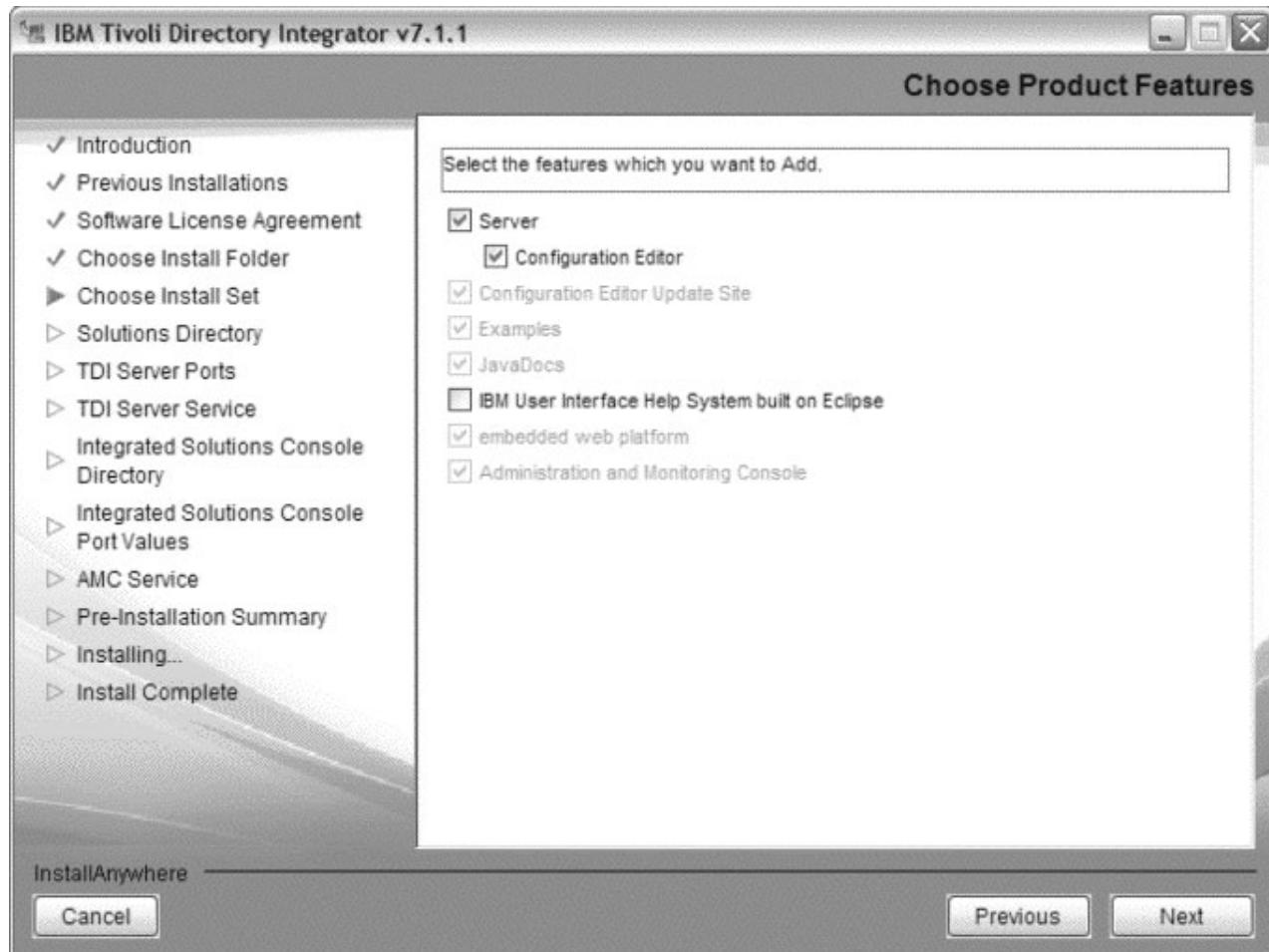
You are not able to choose the **Add Feature** button if there is no TDI 7.1.1 instance available.

You are not able to choose the **Upgrade** button if there are no previous versions of TDI available.

The TDI 7.1.1 drop down is enabled if the **Add Features** button is chosen.

Feature Selection Panel

The next panel in the Add Feature sequence will be the Feature Selection panel, with the already installed features selected and grayed out.



At this point you can add any additional features you choose.

You are not allowed to remove features.

From this point the panel flow matches the new install flow. Panels related to already-installed features will, however, be skipped.

If you select Configuration Editor, Server will also automatically be selected. Also if both features are selected and you deselect Server, then Configuration Editor will also be deselected.

Missing embedded web platform pre-req Panel (i5/OS only)

TDI Solutions Directory Panel

Register Server as a System Service Panel

TDI AMC Deployment Panel

Register AMC as a service Panel

Pre-Install Summary Panel

Installation Progress Panel

Installation Complete Panel

Migration Panel flow

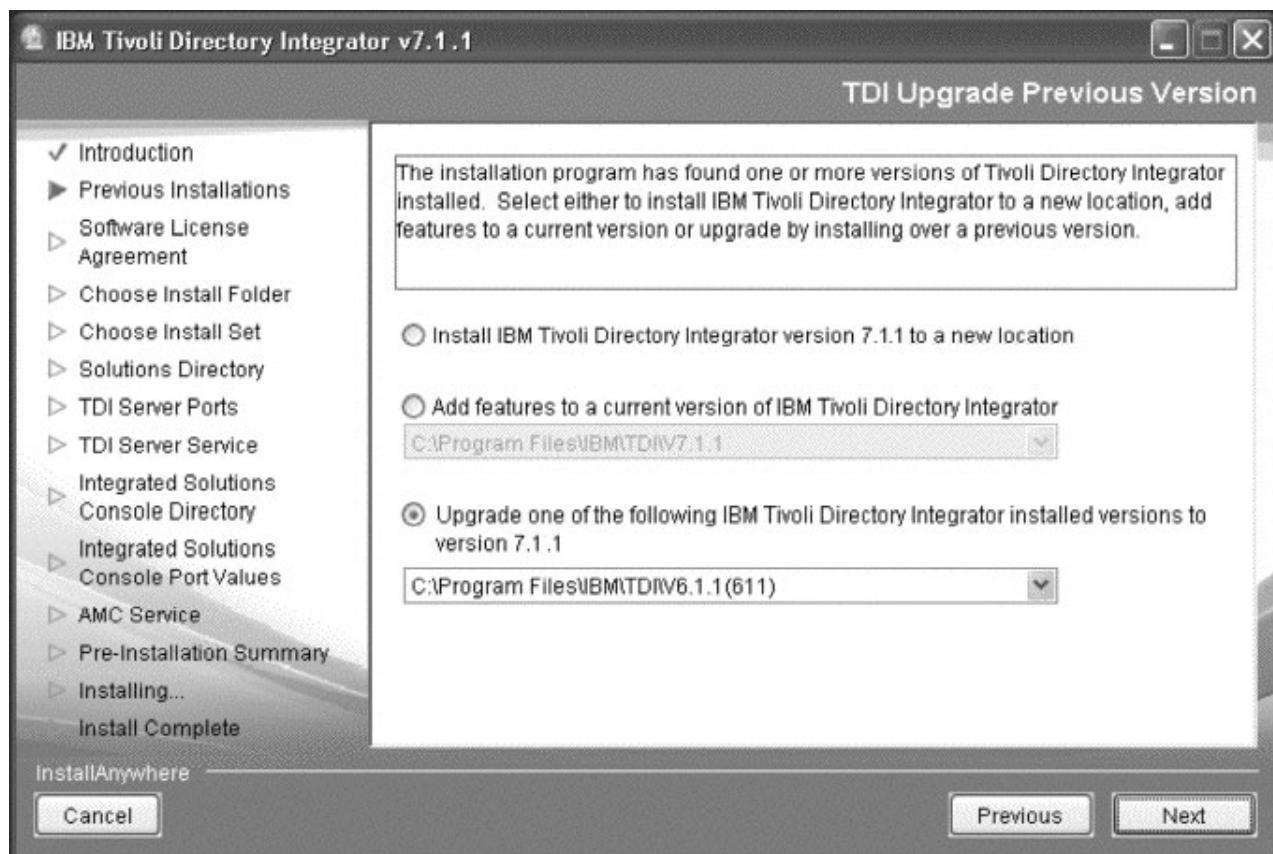
The Migration flow is similar to the new install flow. Only the unique panels will be shown here.

Pre-Initialization Panel

The Welcome Panel

Upgrade Panel

After the Welcome panel and the Previous TDI information panel, if there is an instance of TDI already installed on the box, you will see this panel:



You are not able to choose the Add Feature button if there is no TDI 7.1.1 instance available.

You are not able to choose the Upgrade button if there are no previous versions of TDI available.

The previous TDI version drop down is enabled if the Upgrade button is chosen.

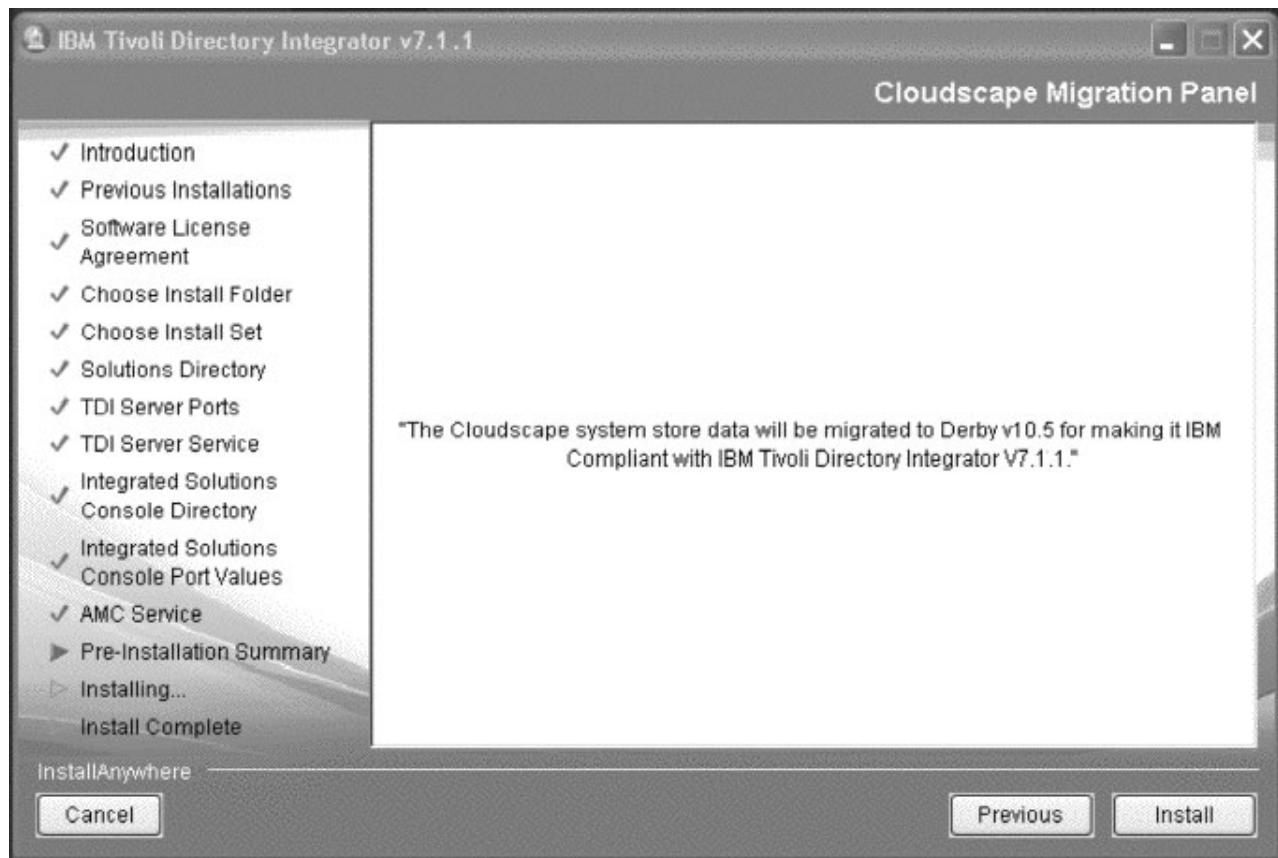
TDI 6.0 Upgrade Flow

If you choose to upgrade a TDI 6.0 instance, the next panel shown will be the License panel and after accepting the license, the Install Type panel. If the customer chooses a Custom install, then the normal Feature list will be presented.

If you choose to install the Server feature, the TDI Solutions Directory panel will be skipped; the value will be obtained from the TDI 6.0 install.

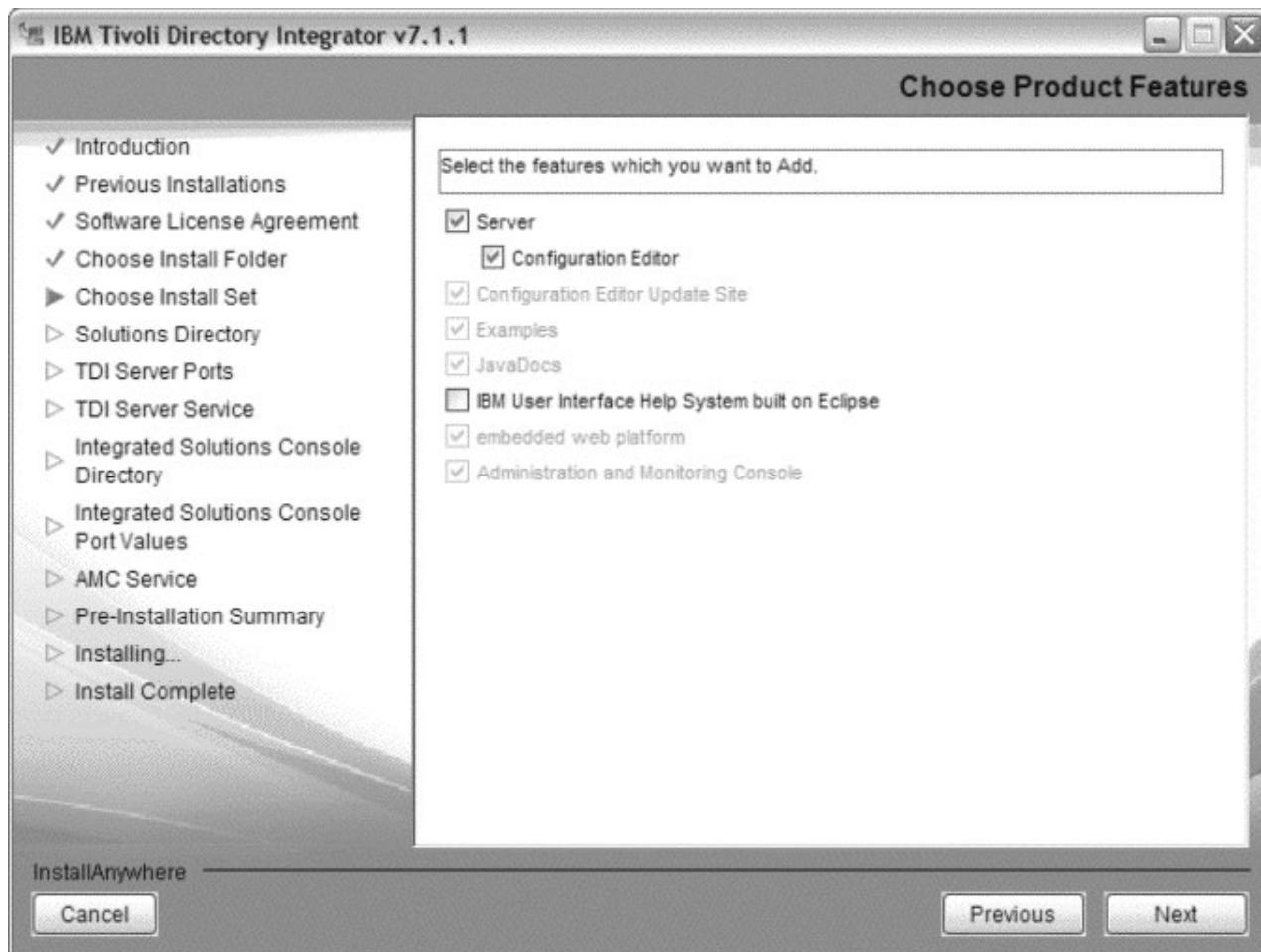
If you choose to install AMC, you will see the normal panels.

If the installer detects that the TDI 6.0 instance used Cloudscape, you will see this panel:



TDI 6.1.x, 7.0 Upgrade Flow

If you choose to upgrade TDI 6.1, 6.1.1 or 7.0, the next panel you will see will be the License panel and after accepting the license, the Feature selection panel:



It will show those features (selected) that were already installed in the previous version, and allow you to add new ones. You are not allowed to disable features that were previously installed.

If the Server was previously installed, the TDI Solutions Directory panel will be skipped. The installer will use the value from the previous installation.

If AMC was previously installed, you will still see the Choose ISC panel, in a TDI 6.x.x migration.

The rest of the panel flow is the same as a new install. In a TDI 7.0 migration, there is one new feature: **Register server as system service**, this panel is only shown after the feature selection panel.

If you select Configuration Editor, Server will also automatically be selected. Also if both features are selected and you deselect Server, then Configuration Editor will also be deselected.

Installing using the command line

The following command line options are supported by the Tivoli Directory Integrator 7.1.1 installer:

-i Sets the installer interface mode: silent, console or gui.
`install_tdiv711_win_x86.exe -f Response File Name -i silent`

`install_tdiv711_win_x86.exe -i console`

-f Sets the location of a response file (installer.properties file) for the installer to use.
`install_tdiv711_win_x86.exe -f installer.properties`

This path can be absolute or relative. (Relative paths are relative to the location of the installer.)

- r Creates a response file.

```
install_tdiv711_win_x86.exe -r myinstaller.properties
```

Note: The Tivoli Directory Integrator 7.1.1 installer creates the tdi_respsfile711.txt response file in the system's temporary files directory, even if -r option is not specified. For example:

- On Windows platform, the response file is created in the C:\DOCUME^1\ADMINI^1\LOCALS^1\Temp directory.
- On non-Windows platform, the response file is created in the /tmp directory.

The *TDI_install_dir/examples/install* directory contains example response files for various installation and uninstallation scenarios.

- D Passes custom command-line arguments.

```
install_tdiv711_win_x86.exe -Dmyvar=myvalue
```

- l Uses the specified language code (and optional country code) to set the locale for the InstallAnywhere installer.

```
install_tdiv711_win_x86.exe -l en
```

```
install_tdiv711_win_x86.exe -l pt_BR
```

The required language code is a two-character (commonly lowercase) code defined by the ISO-639 standard. InstallAnywhere accepts both old (iw, ji, and in) and new (he, yi, and id) language codes.

The optional country code is a two-character (commonly uppercase) code defined by the ISO-3166 standard.

Locale options are only respected if the installer includes localizations for the locale you specify.

- ? Shows help for the InstallAnywhere installer.

On Windows, -help only works from the console launcher. Make sure to set the LaunchAnywhere to **Console** on the Windows tab of the **Project > Platforms** subtask. (For an installed LaunchAnywhere to provide this information, you need to make sure it is explicitly set to Console Launcher on the action.)

The following command line option is unique to the Tivoli Directory Integrator installation Wizard:

LAX_VM

The LAX_VM parameter is used to boot the installer from Java virtual machine, which is installed on the system.

You need to specify absolute path of the Java executable file that resides in the Java bin directory. For example,

```
install_tdiv711_win_x86.exe LAX_VM "Java_DIR/jre/bin/java.exe"
```

Use only the space characters between the arguments.

Note: Make sure that you use the absolute path of IBM JRE version 1.5 and above, as parameter value. The Tivoli Directory Integrator 7.1.1 installer may not work correctly with other JREs.

-D\$TDI_BACKUP\$="true"

This parameter should only be passed in on an uninstallation. This parameter is provided for future migration considerations; for example:

```
TDI_install_dir\uninst\uninstaller.exe -D$TDI_BACKUP$="true"
```

This instructs the uninstaller to run the *TDI_install_dir/bin/tdiBackup.bat(.sh)* script, which in turn will cause a directory *TDI_install_dir/backup_tdi* to be created. A backup of a number of files particular to your installation will be stored into this directory, including your global properties files, global certificates and the like.

-D\$TDI_SKIP_VERSION_CHECK\$="true"

This parameter will cause the installer to skip any previous version checks. This essentially disables any migration from previous releases.

In a silent install, if this skip option is chosen and the install directory is same as an earlier installation of Tivoli Directory Integrator V7.1.1, it will cause the installer to stop.

-D\$TDI_NOSHORTCUTS\$="true"

This parameter is used to stop the installer from creating any shortcuts to the uninstaller, CE, or AMC.

Note: The Installation and Uninstallation Launcher commands will not work on i5/OS. Tivoli Directory Integrator does not use IA-generated launchers on i5/OS, but instead, uses shell scripts. Since launchers are not used, the launcher commands are not applicable. The run-time command-line options will work.

Temporary file space usage during installation

During installation, the installer may use a substantial amount of temporary file space in order to stage files. If your system is constrained in this regard, errors during installation might occur.

UNIX/Linux systems typically use /tmp or /var/tmp as temporary files storage, whereas on Windows, the temporary file storage area is found in the location pointed to by the environment variable TEMP.

InstallAnywhere installers can be instructed to redirect their temporary file usage by setting the environment variable IATEMPDIR before starting the installer. For example, on UNIX:

```
export IATEMPDIR=/opt/IBM/TDI/temp
```

Then, start your console mode installers from the session in which you have set the IATEMPDIR variable.

Performing a silent install

To perform a silent installation you must first generate a response file. To generate this file, perform a non-silent installation with the -r option specified, for example:

```
install_tdiv711_win_x86.exe -r Response File Name
```

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

Note: The directory *TDI_install_dir/examples/install* contains a number of example response files for various installation and uninstallation scenarios.

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

```
install_tdiv711_win_x86.exe -i silent -f Response File Name
```

Note: The examples in this document use the Windows platform installation executable file. See “Launching the appropriate installer” on page 9 for a list of executable file names for each supported platform.

Post-installation steps

CE Update Site

If the CE Update site was installed, you now have to manually deploy into Eclipse. See the section entitled “Installing or Updating using the Eclipse Update Manager” on page 48 for more information.

Plug-ins

If any of the password synchronization plug-ins were installed, see the *IBM Tivoli Directory Integrator V7.1.1 Password Synchronization Plug-ins Guide* for information on how to deploy the plug-in code.

Administration and Monitoring Console (AMC)

General information:

- For more information on AMC, see Chapter 16, “Administration and Monitoring,” on page 213.
- When you are ready to log into the console, browse to <http://hostname:port/ibm/console>. For more information, see section “Log in and logout of the console” on page 229.
- For more information on adding users and user roles, see section “Console user authority” on page 218.

Bundled embedded web platform deployment:

- If you installed AMC with the bundled embedded web platform and are ready to use AMC, you will need to run the commands to start AMC and Action Manager (AM) before you can log into the ISC console. For more information, see section “Starting the Administration and Monitoring Console and Action Manager and logging in” on page 214.

Note: On Windows, a shortcut to the launchAMC.html file is created in the start menu under Program Files.

- By default, the user who installs Tivoli Directory Integrator is the only one with access to log in to the console.

Customer or deferred deployment:

- If you chose a custom Tivoli Integrated Portal (ISC embedded) to deploy AMC to, and are now ready to deploy, see “Deploying AMC to a custom ISC SE or Tivoli Integrated Portal (ISC embedded)” on page 47. When deploying AMC in such a way the installer does not automatically assign the current user the TDI AMC Admin role. This right needs to be manually authorized by an administrator of the ISC console. This is typically done using the **Users and Groups -> Administrative User Roles** panel of the Tivoli Integrated Portal (ISC embedded) console. Alternatively this role could be assigned using the `setAMCRoles` command.
- If you chose to defer deployment of AMC into an ISC and are ready to do so, see section “Deploying AMC to a custom ISC SE or Tivoli Integrated Portal (ISC embedded)” on page 47.

Note: If you have done a custom ISC SE/AE deployment then at a minimum you will need to ensure that AM is started after you start the ISC SE/AE that AMC was installed into.

Documentation

The documentation system used by Tivoli Directory Integrator is the IBM Eclipse Help System. After you have done a default installation, this means that IBM Tivoli Directory Integrator 7.1.1 documentation is made available to you online, on the Web in the form of an Infocenter hosted by IBM. You may, however, choose to deploy the documentation locally. For more information, see “Installing local Help files” on page 46.

If you are new to Tivoli Directory Integrator, we recommend that you read and step through the *IBM Tivoli Directory Integrator V7.1.1 Getting Started* in order to get used to the concepts used.

If you have used earlier versions of Tivoli Directory Integrator, then chapter 3 of the *IBM Tivoli Directory Integrator V7.1.1 Users Guide* will be very beneficial to you in order to understand the new IDE framework and layout. It will also explain how you can import and open your existing configurations; and how the Server still uses the Config model at runtime.

Migration

If you have used an earlier version of Tivoli Directory Integrator, then you will most likely need to migrate certain aspects of your previous deployment. More information on what to do in this case can be found under Chapter 5, "Migrating," on page 63.

Attention: Support for running the Configuration Editor (the GUI for developing solutions in IBM Tivoli Directory Integrator 7.1.1) has changed. The Configuration Editor (CE) is not supported on the following platforms:

- z/OS
- Linux 390
- Linux PPC
- HP-UX Integrity

If you wish to develop solutions for these platforms, you should use the Remote Configuration Editor functionality, meaning that you run the CE on a supported platform, while in contact with a Tivoli Directory Integrator Server on one of the aforementioned platforms. See "Using the Remote Configuration Editor" on page 131 for details.

Installing local Help files

The IBM Tivoli Directory Integrator installer does not contain any user documentation, other than the Java API documentation, which can be displayed by selecting the **Help -> Welcome** screen, **JavaDocs** link in the Configuration Editor. IBM provides the user documentation in online form in an information center, at http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.1.1/welcome.htm.

IBM Tivoli Directory Integrator is equipped with code¹ to provide you with context-dependent online help that you can launch from the Configuration Editor (CE). By default, this code resolves the documentation from the online information center 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:

- The code to handle the documentation files, the IBM User Interface Help System built on Eclipse, is not installed by default. In order to install the Help system, you will need to do a custom install, and install the Help system feature into your existing Tivoli Directory Integrator installation.
- All the manuals are stored together in one compressed directory, which when uncompressed contains an Eclipse *Document plug-in*.
- All the manuals can be downloaded, in their compressed form, from the Tivoli Directory Integrator documentation site, at http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.1.1/welcome.htm
- The entire documentation package, *di_plug-ins-7.1.1.zip*, should be uncompressed into the right place: *TDI_install_dir/ibm_help/eclipse/plug-ins* folder (or uncompress somewhere else, and move into the right place). The package contains the actual Tivoli Directory Integrator documentation in *com.ibm.IBMDI.doc_7.1.1*, alongside a number of other directories whose names end in .doc; all of those directories should be at the same aforementioned *plug-ins* 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 Solutions Directory. By default, this points to the online information center, 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/tivihelp/index.jsp?topic=
```

1. The help system is powered by Eclipse™ technology. (<http://www.eclipse.org>)

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=
```

then next time you run the CE and launch Help, it starts a local task to serve the documentation, from the content in the *plug-ins* directory.

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

After you install the documentation in the *plug-ins* directory as outlined above, you can also decide to host the documentation on that computer for other installations of IBM Tivoli Directory Integrator in your environment. In the *TDI_install_dir/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, the script starts an information center 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". On those clients that are trying to access this information center, 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 information center. In addition, in order for AMC to find this information center, you must update the parameters *InfoCenterHostname* and *InfoCenterPort* attributes in its 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 information center.

help_start.bat or help_start.sh

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

help_stop.bat or help_stop.sh

Stop the local Java task that was started by WebSphere_help_start.

Deploying AMC to a custom ISC SE or Tivoli Integrated Portal (ISC embedded)

If you chose to defer deployment of AMC to ISC, and are now ready to deploy, follow these steps:

- Execute the following scripts:

```
TDI_install_dir/bin/setISCHome.bat(sh) ISC location  
TDI_install_dir/bin/amc/install.bat(sh)  
TDI_install_dir/bin/amc/setAMCRoles.bat(sh) username
```

Notes:

1. Calling the *setAMCRoles* script is optional for both SE and AE. If executed, *username* should be an already existing one on the ISC/WAS environment. As an alternative, you can use the ISC console ("Console User Authority" panel specifically) to manually assign one of the roles that came with AMC - "TDI AMC Admin" and "TDI AMC User" to a user.
2. See the section "AMC in the Integrated Solutions Console" on page 218 for more information on AMC roles.

- Alter the amc.properties file so the lines specifying am.api.port and amc.help.port have appropriate port values. For ISC SE this file is located in *ISC location/runtime/isc/eclipse/plugins/AMC_7.1.1.0/* and for Tivoli Integrated Portal (ISC embedded) this file is located in *ISC location/systemApps/* isclite.ear/tdiamc.war

If you chose a custom Tivoli Integrated Portal (ISC embedded) to deploy AMC to, and are now ready to deploy, follow this step:

- Execute the following script:

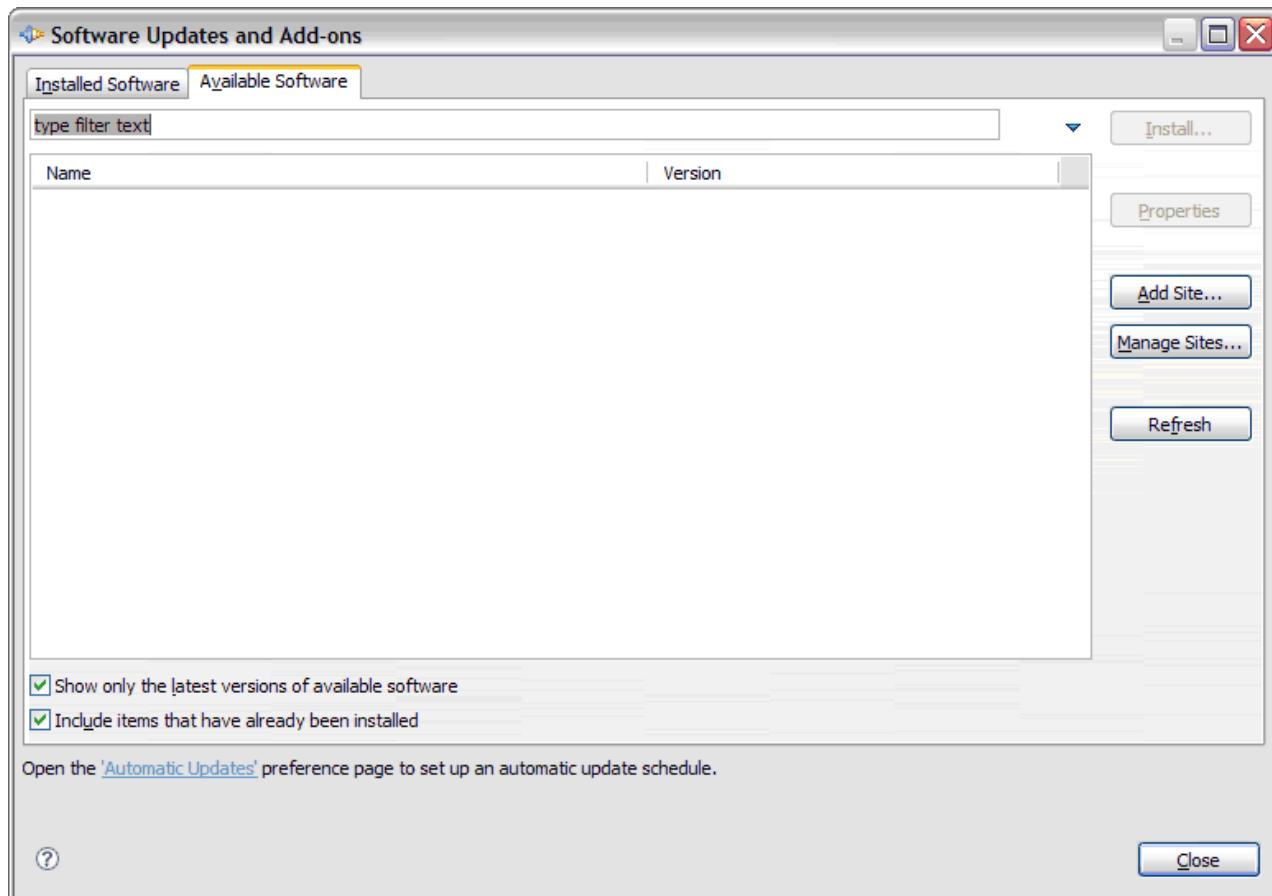
```
TDI_install_dir/bin/amc/setAMCRoles.bat(sh) username
```

Notes:

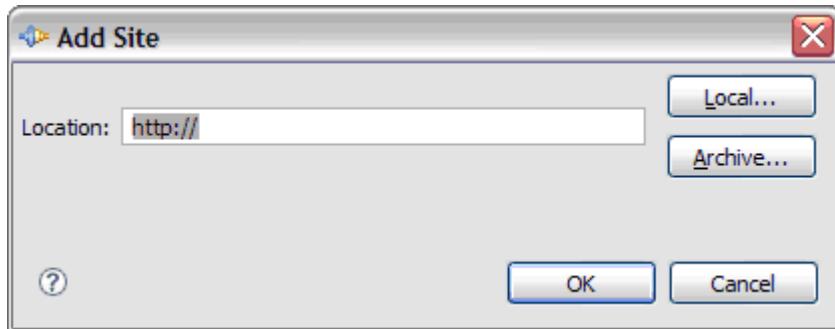
- Calling the setAMCRoles script is optional for both SE and AE. If executed, *username* should be an already existing one on the ISC/WAS environment. As an alternative, you can use the ISC console ("Console User Authority" panel specifically) to manually assign one of the roles that came with AMC - "TDI AMC Admin" and "TDI AMC User" to a user.
- See the section "AMC in the Integrated Solutions Console" on page 218 for more information on AMC roles.

Installing or Updating using the Eclipse Update Manager

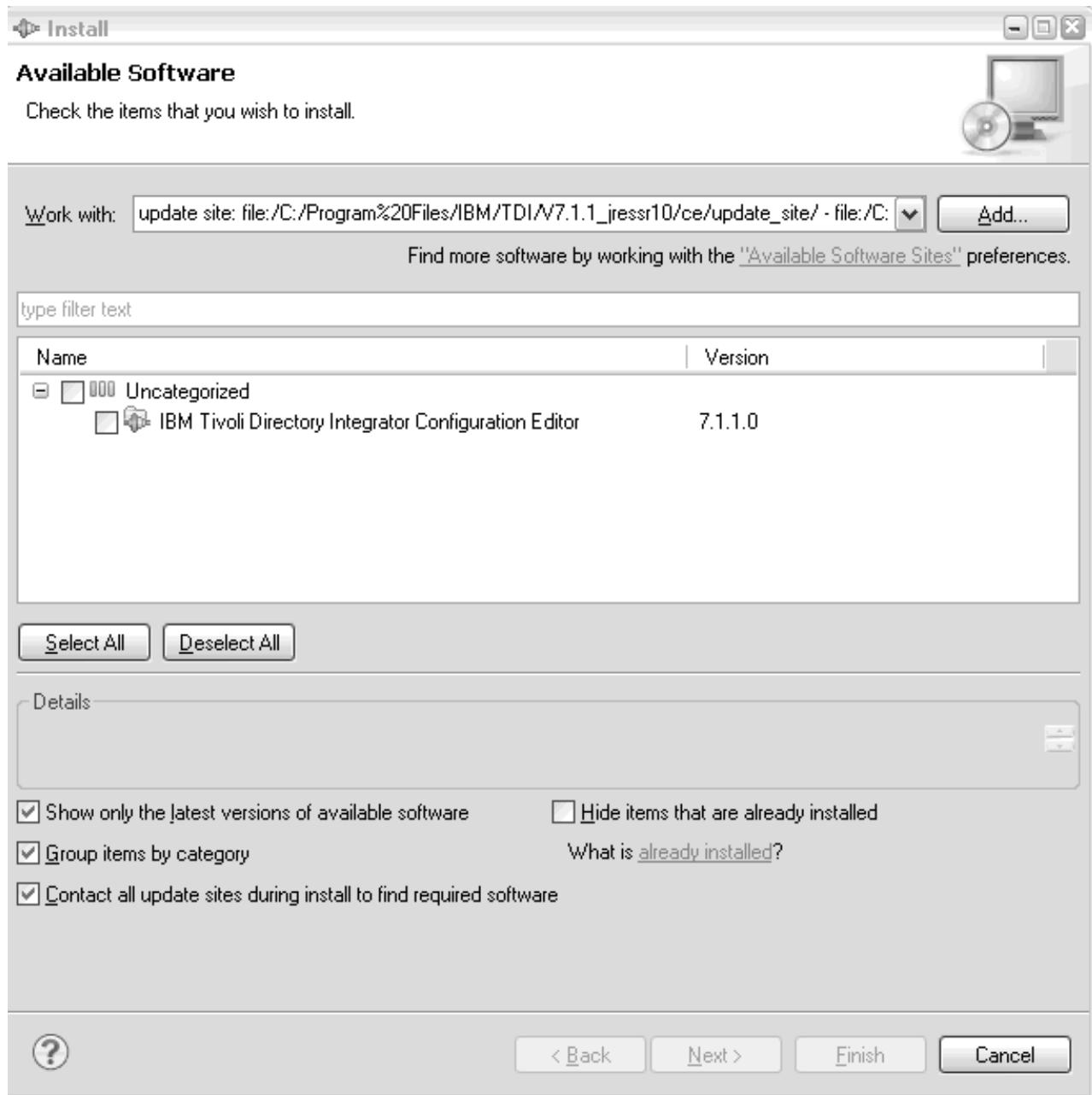
The Tivoli Directory Integrator Rich Client Platform contains a complete runtime environment to run the Tivoli Directory Integrator CE. However, it is possible to install the Tivoli Directory Integrator Eclipse plug-in into an existing Eclipse installation. This is done using the Eclipse Update Manager. In Eclipse, open the Eclipse Update Manager through the **Help** menu.



Before you have the Tivoli Directory Integrator plug-in installed you will want to add a new update site. Choose the **Add Site...** button and specify the location of the update site.



Depending on the location of the update site choose the appropriate action. In this example we choose a directory on the local file system. Using the **Local** button you are prompted to choose a directory which is then filled into the location input field. When you press **OK** the new update site and updates should be available:



Check the plug-ins you want to install and press **Install**. As the software update manager updates your installation you may be prompted to confirm the installation and you are also usually encouraged to restart the workbench after installation. After installation is complete you should see Tivoli Directory Integrator in the **Installed Software** tab.

Post-installation steps

When the CE is installed as a plug-in in another Eclipse installation like in the procedure described above, a number of specific properties must be set to include the *TDI loader*. The Tivoli Directory Integrator loader is an org.eclipse.osgi fragment that provides class loading for the CE.

```
# TDI class loader
osgi.framework.extensions=com.ibm.tdi.loader
osgi.hook.configurators.include=com.ibm.tdi.loader.TDIClassLoaderHook
TDI_HOME_DIR=c\:/Program Files/IBM/TDI/V7.1.1
```

Note that the property TDI_HOME_DIR needs to point to an existing Tivoli Directory Integrator Server installation, since the CE must be able to query many Tivoli Directory Integrator component Java classes in order to work correctly. This installation is also used to create the local development server that the CE uses. The fragment above shows the installation default for Windows; update this to reflect your environment.

There are several ways to set these properties. One is to update the configuration/config.ini file of the Eclipse installation.

Note: After installation and configuration of the CE into Eclipse, you may run into dependency problems. A Technote published about this issue may help you resolve such problems.

Uninstalling

You can uninstall IBM Tivoli Directory Integrator in its entirety, or uninstall only certain components.

Launching the uninstaller

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

Note: Before uninstalling, stop any component that you intend to remove, for example an instance of the Tivoli Directory Integrator Runtime, an AMC service that is running, or a Password Synchronization plug-in. Not stopping running components may cause some files to not be removed (to remain after the uninstallation). On Windows, a restart may be required and the Tivoli Directory Integrator Web Admin (AMC) service may remain in the Services list, requiring manual deletion.

1. Navigate to the Tivoli Directory Integrator _uninst directory, for example:
install_path/_uninst
2. Launch the uninstaller by executing the uninstall executable file.
For Windows platforms, the uninstall executable file is called `uninstaller.exe`. For all other platforms, the uninstall executable file is called `uninstaller.bin`, except for i5/OS where it is called `uninstaller.sh`.
3. You will now enter the “Uninstall Panel flow” on page 32.

Attention: During an uninstallation, a number of directories on the computer are emptied and removed. These are:

- *TDI_install_dir/lwi* - There is the possibility that some files are left over here, or files can get created by the embedded Web platform that the installer doesn't lay down. This directory is deleted on uninstallation.
- *TDI_install_dir/ce/eclipsece/features/com.ibm.tdi.*.jar*
- *TDI_install_dir/ce/eclipsece/plugins/com.ibm.tdi.*.jar*
- *TDI_install_dir/ce/eclipsece/configuration*
- *TDI_install_dir/ce/update_site/features/com.ibm.tdi.*.jar* - If any features have been added that match this wildcard, they will be deleted.
- *TDI_install_dir/ce/update_site/plugins/com.ibm.tdi.*.jar* - same as previous.
- *TDI_install_dir/maintenance/BACKUP* - This directory may be created by the update installer.
- *TDI_install_dir/_uninst/** - This is needed because of the way the uninstallation on i5OS happens. It will be deleted regardless.

Anything you may have put into these directories yourself that matches any of these criteria, will be removed as well during an uninstallation.

Performing a silent uninstallation

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

```
TDI_install_dir/_uninst/uninstaller.exe -r UninstallResponseFileName
```

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

Note: The directory `TDI_install_dir/examples/install` contains a number of example response files for various installation or uninstallation scenarios.

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

```
TDI_install_dir/_uninst/uninstaller.exe -f UninstallResponseFileName
```

Note: The examples in this document use the Windows platform uninstallation executable file.

Default installation locations

IBM Tivoli Directory Integrator installs to the following default locations:

Windows platforms

C:\Program Files\IBM\TDI\V7.1.1

Linux and UNIX platforms (AIX, HP-UX, Solaris)

/opt/IBM/TDI/V7.1.1

i5/OS

/QIBM/ProdData/IBM/TDI/V7.1.1

Chapter 3. Update Installer

The Tivoli Directory Integrator Update Installer is an installer included with updates to existing Tivoli Directory Integrator installations; it is used whenever you need to perform maintenance in order to install fixes.

The regular installer lays down a file named .registry in the install directory that represents the current level of installed components. A script named tdiSetBackupDir.bat or tdiSetBackupDir.sh is created in the bin directory of the installation that sets the location of the backup directory; this will be a directory named BACKUP in the maintenance directory by default. You can change the backup location by running the tdiSetBackupDir script. So for example, if a fix is named "ifix1", backup files and directories would be under *install dir/maintenance/BACKUP/ifix1* in this scenario. The update installer will harvest the name of the backup directory when performing maintenance. The user performing maintenance to Tivoli Directory Integrator should have write permission for the install and backup directories. You should also be aware that during a complete uninstallation, the uninstaller will attempt to delete the default backup directory.

The regular installer also handles maintenance of the .registry file during uninstalling and adding features. When performing a full uninstallation, the .registry file will be deleted along with the other files. When performing a partial uninstallation, only the components being uninstalled will be removed from the registry file, and when adding features, the .registry file will be updated to contain the newly installed features. After adding a feature, you should immediately install all of the fixes that are currently applied. While installing the fix pack, if an error occurs such as low disk space, error during system command execution, and so on, the fix pack is rolled back and the .registry file is updated to the most recent fix applied level, if any. Installing a fix that has previously been applied will only update the newly added features.

The update installer will be comprised of several Java files, but to avoid you having to specify the java executable, a wrapper script is created in the bin directory called applyUpdates.bat(sh). This script will use existing scripts to find the right Java JRE to use and call the underlying code. The script's usage is as follows:

```
applyUpdates -update fix_file.zip [-clean [-silent]]  
applyUpdates -rollback  
applyUpdates -queryreg  
applyUpdates -queryfix fix_file.zip  
applyUpdate -enroll license_file.zip  
applyUpdates -?
```

The options are as follows:

-update

This is used to apply a fix. *fix_file* is the name of the zip file containing the fix; it can be a relative or absolute path. The **-clean** option, which is only available for a fixpack, will erase all of the files that have been backed up prior to applying the current fixpack. You will be prompted to ascertain you want to delete the old data. The **-silent** option suppresses the confirmation prompt. In the event that a fixpack is being reapplied, for example if new features have been added that need the fixpack, the **-clean** option will be ignored. Another thing to note about the **-clean** option is that cleaning the backup directories will limit the ability to rollback to a single level.

During fix pack installation, if an error occurs due to low disk space, error in execution of a system command, and so on, the fix pack installation is rolled back to the most recent fix applied level, if any. You need not use the **-rollback** option manually in case of any failure during installation of the fix pack.

-rollback

This is used to rollback to the state Tivoli Directory Integrator was in before the most recent fix was applied.

-queryreg

This will show the features that are in the current installation as well as all of the fixes applied.

Example output:

```
Information from .registry file in: C:\Program Files\IBM\TDI\V7.1.1
Edition: Identity
Level: 7.1.1.1

Fixes Applied
=====
None

Components Installed
=====
BASE
SERVER
CE
CE UPDATE
JAVADOCS
EXAMPLES
IEHS
EMBEDDED WEB PLATFORM
AMC
    Deferred: false
PLUGINS
```

-queryfix

This will show information about the fix contained in fix_file.zip.

Example output:

```
Information from fix file: C:\fixes\TDI-7.1.1s-TESTFP0001.zip
Name: fixpack1
```

```
Minimum level required to apply fix: 7.1.1.0
Maximum level allowed to apply fix: 7.1.1.1
```

```
Prereq
=====
None
```

```
Components Affected
=====
BASE
CE
EXAMPLES
```

The zip file representing a fix (fix_file.zip in the examples above) will contain a manifest file named .manifest which contains information about applying the fix.

-enroll

This option is used to register an empty, trial, or full license. You can also use this option to upgrade the product from a trial version to a full version. However, this option is used by all of the installers. The license to be enrolled is contained in the zip file and is passed as an argument.

To update a license for upgrading the product from a trial version to full version:

```
applyupdates -enroll <license_file .zip>
```

Note: The <license_file .zip> file can be obtained from IBM sales / support team.

Perform the following steps if TDI server is not starting because of the already enrolled license and due to hardware failure:

1. Take a backup of the <tdi-home>.registry file.
2. In the .registry file, remove the value defined in the license tag. For example,

```
<LICENSE>
  Full
</LICENSE>
```

3. Remove the existing nodelock file from <tdi-home>\license directory.
4. Execute the applyUpdates command.

```
<tdi-home>\bin\applyUpdates -enroll <lumfile.zip>
```

Depending on the number of licenses that are included in the zip file, the resulting message indicates the licenses applied as shown in the following example.

```
./applyUpdates.sh -enroll /tmp/TDI_LUM_FULL.zip
CTGDK0059I Trial license successfully enrolled.
CTGDK0062I Full license successfully enrolled.
```

Note: The lum_file.zip file is deleted after the command is executed.

-? This is for usage information.

The .registry file

Inside the install directory will be a file named .registry. This file represents the level of all Tivoli Directory Integrator components currently installed on the system in this particular install directory. This file is initially created by the installer based on the options chosen at install time.

When a fix is installed, the backed up files are stored in a directory with the name of the fix inside the backup directory. If the fix pack is installed successfully, additional entries will be made to the .registry file that represent the changes made to components by a fix. There will be a FIXES section of the .registry file that represents the fixes that have been applied, and each component will have entries representing which fixes have been applied that have altered them. However, if the fix pack is failed to install, the .registry file will not be updated and will contain the same entries that were exist before applying the fix pack.

The Update Installer recognizes the following components:

- BASE
- SERVER
- Configuration Editor (CE)
- CE_UPDATE
- JAVADOCS
- EXAMPLES
- IEHS
- embedded Web platform
- Administration and Monitoring Console (AMC)
- PLUGINS

The plug-ins component may require some steps that cannot be performed by the update installer and must be performed manually. If there is something to be changed in any of the pwsync.props files (not very likely), you must perform this manually following steps in the manual_readme.txt file of the fix pack. This must happen after installing the fix pack, but before any of the post-install steps listed below. The readme will warn you that the Update Installer will only update files, which the Installer has put down. Files that are copied by you, need to be updated manually as described in the post-install steps.

Here is a list of steps that you need to perform pre/post installing the fix pack (as noted, these steps should be executed only if you have registered the corresponding Password Synchronizers into target systems):

Windows Password Synchronizer

Pre-install steps: None

Post-install steps (only if the fix pack contains an update of the DLL of the Password Synchronizer):

1. Remove the name of the DLL of the Password Synchronizer from the following registry key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\LSA\Notification Packages The DLL file is named "tdipwflt" on 32-bit Windows and "tdipwflt_64" on 64-bit Windows.
2. Reboot Windows, so that the Local Security Authority (LSA) process unloads the DLL of the Password Synchronizer.
3. Replace the DLL inside the system32 Windows folder with the one from the installation of the Password Synchronizer. The DLL path after installation is either *install_dir/pwd_plugins/windows/tdipwflt.dll* or *install_dir/pwd_plugins/windows/tdipwflt_64.dll* depending on the version of Windows.
4. Add the name of the DLL (without the ".dll" extension) inside the registry key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\LSA\Notification Packages
5. Reboot Windows again, so that the LSA loads the new DLL.

Afterwards the Password Synchronizer should run normally, using the updated files.

IBM Directory Server Password Synchronizer

Pre-install steps:

1. Stop the Directory Server.
2. Stop the Proxy process of the Password Synchronizer using the stopProxy command-line utility. This is necessary because the IBM Directory Server Password Synchronizer does not automatically stop its Proxy when being terminated.

Post-install steps: None.

Sun Directory Server Password Synchronizer

Pre-install steps: Stop the Directory Server.

Post-install steps: None.

PAM Password Synchronizer

Pre-install steps: If possible, avoid any password changes while the update takes place. Otherwise unregister the Password Synchronizer from the PAM configuration file.

Post-install steps: If you have unregistered the Password Synchronizer before the update, register it again; see *IBM Tivoli Directory Integrator V7.1.1 Password Synchronization Plug-ins Guide* for more information.

Domino® Password Synchronizer

Pre-install steps: None

Post-install steps: follow the post-install instructions from the chapter "Domino HTTP Password Synchronizer" in *IBM Tivoli Directory Integrator V7.1.1 Password Synchronization Plug-ins Guide*, followed by a new setup of the Domino plug-in as described in the section "Deployment on a single Domino Server".

Installing fixes

If a fix file contains a fix for a component and that component is installed on the system, a number of programmed actions will be performed for the individual components.

If there are any manual steps that must be performed outside of the update installer, instructions will be included in the readme file for the fix; in addition see here for additional remarks about Password Plug-ins fixes.

Rollback

During a Rollback, the Update Installer uses information previously laid out during a fix, and files backed up, to restore a previous state.

Troubleshooting

The Update Installer creates a log file named `updateinstaller.log` in the `install_dir/logs` directory. The default level of messages is INFO, but this can be changed by altering the `install_dir/etc/updateinstaller-log4j.properties` file so that DEBUG messages are also logged.

Chapter 4. Supported platforms

Some of the software components in IBM Tivoli Directory Integrator, like the Launchpad and the Administration and Monitoring Console, require a configured Web browser. Supported Web browsers are:

- Microsoft Internet Explorer
- Mozilla Firefox

These are the platforms supported for IBM Tivoli Directory Integrator 7.1.1. For the latest list of platforms supported see: <http://www-1.ibm.com/support/docview.wss?rs=697&uid=swg21579412>.

This URL can also be used to see the platform supported by the Tivoli Directory Integrator plug-ins and the bit support per platform.

Note: On AIX®, Linux on Power, Sun Solaris SPARC, native z/OS and i5/OS, the Tivoli Directory Integrator Server, Configuration Editor (CE) and other components 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.

- Microsoft Windows Intel x86
 - Windows 7 Professional, Enterprise and Ultimate Edition (supported for application design, development, and testing only; no support for production use) (32-bit supported)
 - Windows 2003 Standard, Enterprise and Datacenter Edition (32-bit supported)
 - Windows 2003 R2 Standard, Enterprise and Datacenter Edition (32-bit supported)
 - Windows 2008 Standard, Enterprise and Datacenter Edition (32-bit supported)
 - Windows 2008 R2 Standard, Enterprise and Datacenter Edition (32-bit supported)
- Microsoft Windows Intel x86-64
 - Windows 7 Professional, Enterprise and Ultimate Edition (supported for application design, development, and testing only; no support for production use) (64-bit supported)
 - Windows 2003 Standard, Enterprise and Datacenter Edition (64-bit supported)
 - Windows 2003 R2 Standard, Enterprise and Datacenter Edition (64-bit supported)
 - Windows 2008 Standard, Enterprise and Datacenter Edition (64-bit supported)
 - Windows 2008 R2 Standard, Enterprise and Datacenter Edition (64-bit supported)
- AIX
 - AIX 5L 5.3 (32/64-bit supported) (Recommended Maintenance package 5300-03 is required)
 - AIX 6.1 (11/2007) (32/64-bit supported) (WPARs are supported)
 - AIX 7.1 (PPC_64)
- Sun Solaris SPARC
 - Solaris 9 (32/64-bit supported)
 - Solaris 10 (32/64-bit supported) (Zones are supported)
 - Solaris 11 (32/64-bit supported)
- Sun Solaris Opteron
 - Solaris 10 (64-bit supported)
 - Solaris 11 (64-bit supported)
- HP-UX Integrity
 - HP-UX11iv2 (11.23) (64-bit supported)
 - HP-UX11iv3 (64-bit supported)

Note: From Tivoli Directory Integrator V7.1, no versions of HP-UX PA-RISC are supported anymore.

- Linux Intel x86
 - RedHat Enterprise Linux ES/AS 4.0 (32-bit supported)
 - RedHat Enterprise Linux ES/AS 5.0 (32-bit supported)
 - RedHat Enterprise Linux ES/AS 6.0 (32-bit supported)
 - SLES 10 (32-bit supported)
 - SLES 11 (32-bit supported)
 - Red Flag Data Center 5.0 SP1 / Asianix 2.0 SP1 (32-bit supported)

Note: A prerequisite on RedHat Enterprise Linux ES/AS 4.0 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.

For RedHat Enterprise Linux ES/AS 5.0, compat-libstdc++ and libXp are provided on 2nd installation CD for RHEL5 are required.

For RedHat Enterprise Linux ES/AS 6.0, library package glibc, compat-libstdc++, nss-softokn-freebl, and libgcc are required.

- Linux Intel x86-64
 - RedHat Enterprise Linux ES/AS 4.0 (64-bit supported)
 - RedHat Enterprise Linux ES/AS 5.0 (64-bit supported)
 - RedHat Enterprise Linux ES/AS 6.0 (64-bit supported)
 - SLES 10 (64-bit supported)
 - SLES 11 (64-bit supported)

Note: A prerequisite on RedHat Enterprise Linux ES/AS 4.0 is that library package compat-libstdc++-296-2.96-132.7.2 or above is installed for the Installer to work.

For RedHat Enterprise Linux ES/AS 5.0, compat-libstdc++ and libXp are provided on 3rd installation CD for RHEL5 are required.

For RedHat Enterprise Linux ES/AS 6.0, library package glibc, compat-libstdc++, nss-softokn-freebl, and libgcc are required.

- Linux on Power (pSeries®, iSeries®, OpenPower® and JS20 Blades)
 - RedHat Enterprise Linux ES/AS 4.0 (32/64-bit supported)
 - RedHat Enterprise Linux ES/AS 5.0 (32/64-bit supported)
 - RedHat Enterprise Linux ES/AS 6.0 (32/64-bit supported)
 - SLES 10 (32/64-bit supported)
 - SLES 11 (32/64-bit 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.

For RedHat Enterprise Linux ES/AS 5.0, compat-libstdc++ and libXp are required.

For RedHat Enterprise Linux ES/AS 6.0, library package glibc, compat-libstdc++, nss-softokn-freebl, and libgcc are required.

- Linux s/390 and zSeries®
 - RedHat Enterprise Linux ES/AS 4.0 (64-bit supported)
 - RedHat Enterprise Linux ES/AS 5.0 (64-bit supported)
 - RedHat Enterprise Linux ES/AS 6.0 (64-bit supported)
 - SLES 10 (64-bit supported)

- SLES 11 (64-bit 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.

For RedHat Enterprise Linux ES/AS 5.0, compat-libstdc++ and libXp are required.

For RedHat Enterprise Linux ES/AS 6.0, library package glibc, compat-libstdc++, nss-softokn-freebl, and libgcc are required.

- Native s/390 and zSeries
 - z/OS 1.10, 1.11, and 1.12 (31/64-bit supported) – with limitations; refer to Chapter 20, “z/OS environment Support,” on page 305
- iSeries
 - i5/OS V6R1 and V7R1 (32/64-bit supported); the CE is not supported on this platform. In addition, the JVM or embedded Web platform to be used is the system-installed one(s); no JVM or embedded Web platform is installed as part of the Tivoli Directory Integrator product install.
The Tivoli Directory Integrator server on i5/OS requires that IBM Technology for Java Virtual Machine (also known as "J2SE 6.0 32-bit JVM" or "IBM J9 VM") be installed on the system
The installation can only be performed locally, as an Administrator, and using a console installation. No GUI installation facility exists for i5/OS.

Notes:

1. "x86" denotes an Intel-architecture 32-bit CPU; "x86-64" denotes an Intel-architecture 64-bit CPU. 64-bit CPUs generally support 32-bit and 64-bit versions of assorted operating systems; 32-bit CPUs only support 32-bit versions.
2. The Tivoli Directory Integrator Config Editor is not supported locally on HP-UX Integrity, Solaris Opteron, Linux on Power, Linux s/390, Native s/390 and zSeries, or i5/OS. The Config Editor will have to be run remotely from a supported operating system.
3. The Administration and Monitoring Console is not supported locally on Native s/390 and zSeries. The Administration and Monitoring Console will have to be run remotely from a supported operating system.
4. The Windows x86 and Linux x86 installers can be used on Windows x86-64 and Linux x86-64 platforms if a 32-bit JRE is needed on these 64-bit platforms. However, the 32-bit installer is not able to install the Password interceptor Plug-ins on a 64-bit platform, as the libraries to support such operations do not exist. If you need both a 32-bit Tivoli Directory Integrator installation and Password Plug-ins, you will need to install a second, 64-bit version of Tivoli Directory Integrator in a different location first, and then use a Custom install to install the Plug-ins using that 64-bit installation.

Virtualization support

Virtualization technology allows a single physical machine to be partitioned into multiple physical or logical partitions with each partition providing the look and feel of an independent operating system. Each partition is called a Virtual Environment. Each Virtual Environment represents a complete system, with processors, memory, networking and other system resources. Examples of some virtualization technologies include Sun Solaris Zones, AIX WPARs, and VMware.

IBM Tivoli Directory Integrator 7.1.1 currently provides support for these virtualization technologies:

- VMware; see the following for specifics: <http://www-01.ibm.com/support/docview.wss?rs=697&context=SSCQGF&dc=DB520&q1=VMWare&uid=wws1e333ce0912f7b152852571f60074d175&loc>
- Sun Solaris Zones (on Solaris SPARC 10)
- AIX WPARs

Chapter 5. Migrating

In the context of Tivoli Directory Integrator, "migrating" can mean a number of things.

- Prepare relevant files (and their contents) to be used in a new location, on the same machine or a different one; or
- prepare relevant files to be used with a new version of the product.

The following table summarizes migration scenarios:

Table 1. Migration scenarios

Source & Destination versions equal?	Source & Destination install paths equal?	Scenario description
no	no	Migrate files to a new version, which is installed in a different location
no	yes	Migrate files to a new version, which will be installed in the same location
yes	no	Migrate files to a different installation of the same version
yes	yes	Restore backed up files to their original location

If you have to both migrate to a new version and a new location, you should do the version upgrade first, because here we will cover location migration only for the current release (7.1.1).

The IBM Tivoli Directory Integrator Installer can assist in migrating from Tivoli Directory Integrator 6.0, 6.1.X, and 7.X to Tivoli Directory Integrator 7.1.1.

Migrate files to a different location

In this section we cover only 7.1.1.

Which files do not need to be modified to be used in another location?

- User configurations, data files (.xml, .xsd, .xsl, .txt ...), key store files (.jks, ...), certificate files (.der, ...), and so forth.

Also consider the implications of section “Maintaining encryption artifacts – keys, certificates, keystores, encrypted files” on page 164.

- Scripts (see this section for exceptions)
- .bat, .sh and .vbs files
- JAR files
- Native binaries - .exe, .dll, .so, and so forth.
- Server API registry file
- Server Stash File
- Derby databases:

For example the default System Store database "TDISysStore" and the default AMC database "tdiamcdb".

Note that you must move the database as a whole (the whole folder). You should not merge the files of two databases.

For more complicated scenarios, transfer data between databases using the JDBC Connector.

- Action Manager property files (located in the *TDI_install_dir/bin/amc/ActionManager* folder)

- Configuration files from the "etc" folder except these:
 - build.properties
 - global.properties
 - updateinstaller-log4j.properties
 - tdisrvctl-log4j.properties
- AMC configuration files:
 - amc.properties
 - amcdbhandler.properties
 - amcdbschema.xml
 - idiamc.sth

Which files need to be modified before they can be used in another location?

In general, location-sensitive files will contain the absolute path of the installation folder in one or more places. These occurrences need to be replaced with the new location path, so that the file becomes relevant to the new location.

Below is a list of the files that need migration and hints about which fields to update. These hints are based on the default content of these files. If you have modified the files, there may be other fields that are also location specific and need to be updated too.

- bin/amc/amcwinservice.ini:
 This is the configuration file for AMC when registered as a Windows service.
 Update the "WorkingDirectory", "StartCommand" and "StopCommand" properties.
- global.properties/solution.properties:
 Update the "com.ibm.di.store.database" property.
 Also consider these properties: "api.config.folder", "systemqueue.jmsdriver.param.mqe.file.ini" and "com.ibm.di.loader.userjars".
 If you migrate the file to an installation that uses a different encryption key, see section "Maintaining encryption artifacts – keys, certificates, keystores, encrypted files" on page 164.
- etc/updateinstaller-log4j.properties:
 Update the "log4j.appenders.Default.file" property.
- etc/tdisrvctl-log4j.properties:
 Update the "log4j.appenders.Default.file" property.
- ibmdiservice.props:
 This is the configuration file for the Server when registered as a Windows service.
 Update the "path", "ibmdirroot" and "jvmRoot" properties.
- pwsync.props:
 These are the configuration files of the Password Synchronizers.
 Update the "proxyStartExe", "logFile", "javaLogFile" and "mqe.file.ini" properties.

Which files should not be used in another location under normal circumstances?

- Certain scripts
 The sole purpose of the existence of these files is to convey location specific data.
 There is virtually nothing else in them, so they would be of little value in another location.
 Consider these scripts from the *TDI_install_dir/bin* folder: "javaHome", "defaultSolDir", "backupDir", "tdiISCHome".

- .reg files
These are used by the Windows Password Synchronizer.
- MQe queue manager files
Although you cannot easily migrate MQe files to another location, you can transfer data from one MQe queue to another using the JMS Connector with an MQe JMS driver.
- CE workspace
To reuse Directory Integrator projects from the Config Editor workspace, export them as Directory Integrator configurations and import them into the new workspace.
- etc/build.properties
This file contains time and version information about the release of the product.

Migrating files that contain encrypted data

See “Maintaining encryption artifacts – keys, certificates, keystores, encrypted files” on page 164.

Migrate files to a newer version

Installer-assisted migration

The installer migrates certain files automatically during upgrade to a newer version. Note that the installer considers only the installation folder of the Directory Integrator.

All solution folders that are different than the installation folder must be migrated manually (or using some of the tools described in section “Tool-assisted migration” on page 66).

Which files does the installer migrate automatically?:

- 6.0 to 7.1
 - global.properties
 - Cloudscape database (if used for System Store) is upgraded to Derby v10.5.3. See “Migrating Cloudscape database to Derby” on page 86.
 - pwsync.props (for each installed Password plug-in)
- 6.1.x to 7.1
 - global.properties
 - the AMC database
 - amc.properties
 - am_config.properties
 - pwsync.props (for each installed Password plug-in)
- 7.0 to 7.1
 - global.properties
 - pwsync.props (for each installed Password plug-in)
- 7.1 to 7.1.1
 - global.properties
 - solution.properties (if exists in default Solution Directory)
 - pwsync.props (for each installed Password plug-in)

Which files need to be migrated manually?:

Everything mentioned in section “Manual migration” on page 66, except those mentioned in its first subsection, Property Files.

Tool-assisted migration

These tools are used by the installer for the installer-assisted migration. You can use them for manual migration.

Property files migration:

- `global.properties`:
Use the "tdimiggbl" tool from *TDI_install_dir/bin*; see section "Migrating global and solution properties files using migration tool" on page 87.
- `amc.properties`:
Use the "tdimigamc" tool from *TDI_install_dir/bin/amc*; see "AMC and AM Command line utilities" on page 251.
- `am_config.properties`:
Use the "tdimigam" tool from *TDI_install_dir/bin/amc*; see "AMC and AM Command line utilities" on page 251.
- `pwsync.props` (for each installed Password plug-in):
Use the "migpwsync" tool from *TDI_install_dir/pwd_plugins/bin*; see "Migrating Password plug-ins properties files using migration tool" on page 88.

AMC database migration:

Use the "backupamcdb"/"restoreamcdb" tools from *TDI_install_dir/bin/amc*; see "AMC and AM Command line utilities" on page 251.

Cloudscape System Store migration (only for 6.0):

See the more detailed instructions in section "Migrating Cloudscape database to Derby" on page 86.

Manual migration

Copy your Config files and any other custom files, including Derby databases from your old installation directory to the new installation directory. Tivoli Directory Integrator 7.1.1 supports a Solution Directory, and we recommend you copy the Config files, property files, Derby databases, and so on, to such a solution directory instead of to the installation directory of Tivoli Directory Integrator version.

Once you have copied the objects referenced above to a new location, you can set out to manually migrate their contents to adapt them for use with IBM Tivoli Directory Integrator 7.1.1 as described in the sections below:

1. Property Files
2. Configurations
3. Customized scripts
4. Added or replaced JAR files in the installation
5. Password Synchronizer configurations

Note: Sandbox data is version-specific; data recorded under any previous version does not play in version 7.1.1.

Property files:

- `global.properties`:
The table below lists which properties have been deleted or changed in Tivoli Directory Integrator 7.1.1:

Table 2. Deleted and changed properties

Old property (pre-v7.0)	New property	Remarks
<code>## Active Correlation Technology engine settings # act.engine.rule.set.file=myrules.acs</code>	*DELETED*	Remove ACT Engine and ACT Connector
<code># Location of directory where the JRE TDI will use is installed com.ibm.di.jvmdir=\$jvmRoot\$</code>	*DELETED*	No longer possible to specify.
<code>com.ibm.di.scriptengine.precompile=true</code>	*DELETED*	No longer possible to specify; the current script engine does not have this functionality.
<code>com.ibm.di.scriptengine.regex=java</code>	*DELETED*	No longer possible to specify - Java syntax is always followed.
<code>ibmjs.options=com.ibm.di.script.ScriptEngineOptions</code>	*DELETED*	Related to previous property; this is no longer a valid option.
<code>com.ibm.di.store.create.checkpoint.store=<multiple statements></code>	*DELETED*	Checkpoint/Restart functionality is removed; any System Store create table statements related to this should be removed too.
<code>com.ibm.di.admin.library.dir</code>	*DELETED*	The current Config Editor does not use this, so no longer possible to specify.
<code>api.remote.on=false</code>	<code>api.remote.on=true</code>	RMI enabled by default in TDI Server – Setting to true since it is enabled by default.
<code>javax.net.ssl.trustStore={protect}-javax.net.ssl.trustStorePassword={javax.net.ssl.trustStoreType}</code>	<code>javax.net.ssl.trustStore=serverapi\testadmin.jks {protect}-javax.net.ssl.trustStorePassword=administrator javax.net.ssl.trustStoreType=jks</code>	RMI enabled by default in TDI Server – empty values replaced by the default truststore.
<code>javax.net.ssl.keyStore={protect}-javax.net.ssl.keyStorePassword={javax.net.ssl.keyStoreType}</code>	<code>javax.net.ssl.keyStore=serverapi\testadmin.jks {protect}-javax.net.ssl.keyStorePassword=administrator javax.net.ssl.keyStoreType=jks</code>	RMI enabled by default in TDI Server – empty values replaced by the default keystore.
<code>com.metamodel.securityTransformation=DES/ECB/NoPadding</code>	<code>com.ibm.di.securityTransformation=DES/ECB/NoPadding</code>	FIPS 140-2 Certification – property name changed.
<code>com.ibm.di.server.keystore=myKeyStore.jks com.ibm.di.server.key.alias=myKeyAlias</code>	<code>api.keystore=myKeyStore.jks api.key.alias=myKeyAlias</code>	Server API keystore properties renamed.
<code>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</code>	<code>#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</code>	The EMBEDDED MODE properties for the System Store have been commented out, since the System Store now runs in Network mode by default. The Installer never makes this change; if you have previously used Cloudscape/Derby in embedded mode you will need to make this change manually.

Table 2. Deleted and changed properties (continued)

Old property (pre-v7.0)	New property	Remarks
<pre>#com.ibm.di.store.database=jdbc:derby://localhost:1527/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.password=APP #com.ibm.di.store.jdbc.startMode=automatic #com.ibm.di.store.jdbc.host=localhost #com.ibm.di.store.jdbc.port=1527 #com.ibm.di.store.jdbc.syslib=true</pre>	<pre>com.ibm.di.store.database=jdbc:derby://localhost:1527/TDISysStore:create=true com.ibm.di.store.jdbc.driver=org.apache.derby.jdbc.ClientDriver com.ibm.di.store.jdbc.urlprefix=jdbc:derby://localhost:1527/ com.ibm.di.store.jdbc.password=APP com.ibm.di.store.jdbc.startMode=automatic com.ibm.di.store.jdbc.host=localhost com.ibm.di.store.jdbc.port=1527 com.ibm.di.store.jdbc.syslib=true</pre>	<p>These are the new, default properties for the System Store in Tivoli Directory Integrator 7.1.1. If you have migrated your installation, you will need to make these changes to your <code>global.properties</code> file as well, if you wish to run the System Store in Networked mode.</p> <p>The new architecture of the Configuration Editor in conjunction with other changes to the development process make that running System Store in embedded mode is very cumbersome. Therefore, we highly recommend that you run in Networked mode.</p>
<code>api.config.folder=\$change\$/configs</code>	<code>api.config.folder=configs</code>	The <code>configs</code> folder is now always local to the Solution Directory.
<pre>##### ## 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 ### 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</pre>	<pre>##### ## 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 ### 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=MQePWStore/pwstore_server.ini</pre>	<p>The System Queue is now enabled by default in Tivoli Directory Integrator 7.1.1 (except on z/OS). Also, the MQe initialization file is now located in a directory subordinate to the Solution Directory.</p>

Table 3. New properties in v7.0

Property	Remarks
<code>com.ibm.di.server.fipsmode.on=false</code>	New property for enabling/disabling FIPS mode was added.
<pre>## To enable the built-in JAAS Authentication mechanism, ## set this property to "[jaas]". api.custom.authentication ## JAAS Authentication properties ## ----- ## java.security.auth.login.config=</pre>	Provide support for JAAS as a Server API Authentication provider; empty property is provided in which you can specify the JAAS Configuration file.
<pre>## Encryption certificate properties com.ibm.di.server.encryption.keystore = <<value of com.ibm.di.server.keystore from 6.1.1 global.properties>> com.ibm.di.server.encryption.key.alias = <<value of com.ibm.di.server.key.alias from 6.1.1 global.properties >> ## Server API keystore passwords {protect}-api.keystore.password= << keystore password from idisrv.sth>> {protect}-api.key.password= << key password from idisrv.sth if present>></pre>	Provide separate configuration options for certificate to be used for PKI Encryption and SSL.
<code>## TDI Logging</code> <code>com.ibm.di.logging.enabled=true</code>	Provide mechanisms to completely disable logging - set to "false" if you want to disable all logging.
<code>derby.connection.requireAuthentication=true</code> <code>derby.authentication.provider=BUILTIN</code> <code>derby.database.defaultConnectionMode=fullAccess</code>	Additional parameters for the System Store (in Derby) in Networked mode.

Table 3. New properties in v7.0 (continued)

Property	Remarks
<pre>##PKCS11 options ##Set the value of following properties to use PKCS11 enabled ## devices to store TDI servers private key / certificate. com.ibm.di.pkcs11cfg=etc\pkcs11.cfg com.ibm.di.server.pkcs11=false com.ibm.di.server.pkcs11.library= com.ibm.di.server.pkcs11.slot= {protect}-com.ibm.di.server.pkcs11.password =PASSWORD</pre>	Support TDI Server's private key/certificate on PKCS 11 compliant crypto devices.
<pre>## Specify the unique ID for the TDI Server ## ----- ## This property helps a client connecting to the TDI server to identify different servers ## running on the same IP and the same port in different time. (Default is blank) com.ibm.di.server.id=</pre>	TDI Server must provide a unique server ID available to remote server clients to detect the server being talked to.
<pre>## Timeout in minutes for loading configuration. api.config.load.timeout=2</pre>	Config initialization and Server API initialization need to be synchronized.
<pre>com.ibm.di.server.encryption.keystoretype = jks com.ibm.di.server.encryption.transformation = RSA</pre>	Symmetric Cipher Support (FIPS 140-2 compliance).
<pre>## Specifies a list of Server notification types, which will be suppressed. ## Notifications of suppressed types will not be propagated by the notifications framework. ## The notification types in the list are separated by spaces. Wildcards may be included. ## Example: ## api.notification.suppress=di.al.* di.ci.start ## The above example will suppress all AssemblyLine related notifications as well as ## notifications for starting a configuration instance. ## If the property is missing or is empty, no notifications will be suppressed. api.notification.suppress=di.server.api.authenticate di.server.api.authorize.*</pre>	Provide TDI Audit Capabilities - Server notification suppression.
<pre>api.audit.on=false</pre>	Provide TDI Audit Capabilities.
<pre>## This property specifies whether LDAP Group authentication is turned on. ## If it is set to 'true', the group membership of the authenticating user ## will be resolved and will be taken into account ## during authorization. ## If it is missing, the default value 'false' is used. api.custom.authentication.ldap.groupsupport= false ## Specifies the name of the attribute of a user in LDAP that contains a list of the groups of which the user is a member. ## It is taken into account only if 'api.custom.authentication.ldap.groupsupport' is set to true. api.custom.authentication.ldap.usermembershipattribute= ## Specifies how groups are named in the membership attribute of a user. ## For example, if the user's membership attribute contains values, ## which correspond to the 'objectSID' attribute of groups, set this property to 'objectSID'. ## If the user's membership attribute contains distinguished names of groups, then set this property to 'dn'. ## The property is required in case 'api.custom.authentication.ldap.groupsupport' is set to true. api.custom.authentication.ldap.usermembershipattributecontent= ## Specifies the name of a group's attribute in LDAP which corresponds to the way the group is named in the TDI User Registry. ## For example, if LDAP groups are addressed in the TDI registry by their common name, then set this property to 'cn'. ## If the User Registry contains the distinguished names of the groups, then set this property to 'dn'. api.custom.authentication.ldap.groupnameattribute= ## Represents the LDAP directory context, where groups will be searched. ## It is required only when LDAP group support is enabled. api.custom.authentication.ldap.groupsearchbase= ## Optional property, which represents a list of space-separated attribute names. ## Specifies attributes which have non-string syntax. ## api.custom.authentication.ldap.binaryattributes=</pre>	Enhance Authorization to support LDAP groups.

Table 4. New properties in v7.1

Property	Remarks
# api.remote.server.ports=8700-8900	<p>Commented out by default; this property is used to configure RMI ports. This is useful in case the default ports conflict with your firewall.</p> <p>The server will use these ports to listen for incoming RMI service requests, in addition to listening on the ports defined by other properties. For outgoing RMI service requests, random port numbers may be used.</p>
<pre>## The properties determine the default bind address and the remote bind address for the Server API. ## * means bind to all network interfaces. The Remote Bind Address overrides the Default one. ## Only one IP address should be set. No hostnames are accepted. ## Mind that the java.rmi.server.hostname property is set implicitly to equal the Remote Bind Address property when used. This will cause the client stubs to create sockets on the specified Remote Bind Address. # com.ibm.di.default.bind.address=* # api.remote.bind.address=*</pre>	<p>Commented out by default; these two properties are used to configure the network interface (hostname or IP address) that the Remote API listens on.</p>
<pre>## Touchpoint Server properties tp.server.on=false tp.server.port=1098 tp.server.config=etc/tp.xml tp.server.auth=false tp.server.auth.realm=Tivoli Directory Integrator Touchpoint Server</pre>	<p>These properties configure the REST interface to TDI connectors using a Service Control Management Protocol (SCMP) based Service.</p>
<pre>## ## Server API client properties ## api.client.ssl.custom.properties.on=true api.client.keystore=serverapi/testadmin.jks {protect}-api.client.keystore.pass=administrator api.client.keystore.type=jks {protect}-api.client.key.pass=administrator api.client.truststore=serverapi/testadmin.jks {protect}-api.client.truststore.pass=administrator api.client.truststore.type=jks</pre>	<p>These properties enable custom SSL properties for Server API clients. If api.client.ssl.custom.properties.on=true, then the api.client.* properties will be used by Server API clients. Otherwise the default javax.net.ssl.* properties will be used.</p>

Table 5. New properties in v7.1.1

Property	Remarks
<pre>## Web container web.server.port=1098 web.server.ssl.on=false web.server.ssl.client.auth.on=false # web.server.session.timeout=300</pre>	<p>These properties are the general settings for REST API and Dashboard. It specifies the port and security settings of the HTTP access point into TDI REST and Dashboard.</p>

Table 5. New properties in v7.1.1 (continued)

Property	Remarks
<pre>## Dashboard properties ## dashboard.on=true dashboard.templates.folder=dashboard/templates ## Dashboard authentication properties ## ## The values for localhost and remotehost can be: ## none: No authentication is required ## deny: All connections denied ## ldap: Authentication is done by logging into an LDAP server and optionally validating group membership ## ## dashboard.ldap.url ## Specify the LDAP host port and optionally a search base (ldap://<host>:<port>[/<search base>]) ## ## dashboard.ldap.url.group ## Specify the LDAP host port and optionally a search base (ldap://<host>:<port>[/<search base>]) ## dashboard.auth=true dashboard.auth.localhost=none dashboard.auth.remote=deny # dashboard.auth.ldap.url=ldap://localhost:389/ou=users,ou=system # dashboard.auth.ldap.url.group=ldap://localhost:389/cn=group1,ou=groups,ou=system</pre>	<p>These properties are specific to the Dashboard. The properties, which are set manually are, “dashboard.on” (enables/disables the Dashboard web application) and the “dashboard.templates.folder” (location of template solutions).</p> <p>All other properties are editable in the Dashboard.</p>
<pre>## REST API ## api.rest.on=true api.rest.auth=false api.rest.auth.realm=Tivoli Directory Integrator REST API api.rest.jmsdriver.name=com.ibm.di.systemqueue.driver.ActiveMQ api.rest.jmsdriver.queue.sender.persistance=false api.rest.jmsdriver.queue.sender.timeToLive=60000 api.rest.jmsdriver.param.jms.broker=vm://localhost?brokerConfig=xbean:etc/activemq.xml # api.rest.jmsdriver.auth.username # api.rest.jmsdriver.auth.password</pre>	<p>These properties configure TDI REST API enablement and security settings. The api.rest.jmsdriver property specifies the JMS queue to use in asynch log messages. Messages are used by the Dashboard API.</p>

Table 6. Deleted/modified in v7.1.1

Old property	New property	Remarks
com.ibm.di.store.database=jdbc:derby: com.ibm.di.hosted52.7.1changes\$TDISysStore;create=true ;name=\$TDIInstallDir1527\$shiro\$TDISysStore;create=true		v7.1 onwards, \$soldir\$ is not replaced with <TDI_install_dir> by default. The directory is updated in runtime inside JVM with current Solution Directory of the user. Thus, TDI System Store is unique for each Solution Directory.
tp.server.port=1098	*DELETED*	This property is redefined as web.server.port=1098.

- amc.properties:

The table below lists which properties have been deleted or changed in Tivoli Directory Integrator 7.1.1:

Table 7. Deleted and changed properties in AMC

Old property (pre-v7.0)	New property	Remarks
AMC.auth	*DELETED*	
monitor.refresh.rate	*DELETED*	The refresh rate for the Monitor Status panel. The rate was specified in minutes.

Table 7. Deleted and changed properties in AMC (continued)

Old property (pre-v7.0)	New property	Remarks
monitor.startup	*DELETED*	To set the Monitor Status panel as the first panel to be seen by the user when (s)he logs in.
LDAPHostName LDAPPort LDAPAdminUID LDAPAdminPwd LDAPServerType LDAPBindID LDAPBindPassword LDAPSuffix LdapUserPrefix LDAPUserSuffix LdapGroupPrefix LDAPGroupSuffix LDAPUserObjectClass LDAPGroupObjectClass LDAPGroupMember LDAPUserFilter LDAPGroupFilter LDAPsearchTimeout LDAPsslEnabled LDAPIgnoreCase	*DELETED*	LDAP Details.
com.ibm.di.amc.jdbc.start.mode	New default value: Automatic	
com.ibm.di.amc.jdbc.host	New default value: Localhost	
com.ibm.di.amc.jdbc.port	New default value: 1528	
com.ibm.di.amc.jdbc.sysibm	New default value: True	

The table below lists which properties have been added in Tivoli Directory Integrator v7.0:

Table 8. New properties in AMC

New property (default)	Remarks
am.logrotate (10)	Used to determine the maximum age of AM log files in days. The log files older than the specified value are deleted. The minimum value is 1 and can be increased up to 2147483647.
amc.session.timeout (20)	Used to determine the maximum time (in minutes) a user is inactive, before the AMC session expires and is automatically logged out of AMC. The value should be a positive integer number.
a1.workEntries.cacheSize (100)	Used by AMC when the AssemblyLine is started in Synchronous mode. The cache size specified here is used for determining the size of the work entries cache.
amc.db.type (derby)	Specifies the database being used by the AMC.
am.api.host (localhost)	Action Manager RMI Details.
am.api.port (13104)	Action Manager RMI Details.
com.ibm.di.server.port.default (1099)	<p>Default port for TDI server.</p> <p>This property can be modified by the TDI installer to have a value different from 1099. When AMC is started for the first time (or if its database was lost), this property is read and its value saved in the newly created AMC database. Later it will be used when AMC connects to the default TDI server instance.</p>

- am_config.properties:

The table below lists which properties have been deleted or changed in Tivoli Directory Integrator 7.1.1:

Table 9. Deleted and changed properties in AM

Old property (pre-v7.0)	New property	Remarks
com.ibm.di.amc.am.serverapi.fail.interval.time=120 com.ibm.di.amc.am.queryProperty.interval.time=600 com.ibm.di.amc.am.healthAL.interval.time=5	*DELETED*	These properties should be commented out as the values for these properties will be configured by the user from AMC while creating each Server API Failure Trigger, On Property trigger and Configuring a Health AL respectively. Hence the properties mentioned in the am-config.properties file will not be used.
com.ibm.di.amc.am.queryAL.interval.time	*DELETED*	
javax.net.ssl.trustStore=\$change\$\${bin/amc/ActionManager/testadmin.jks} javax.net.ssl.keyStore=\$change\$\${bin/amc/ActionManager/testadmin.jks}	javax.net.ssl.trustStore=bin/amc/ActionManager/testadmin.jks javax.net.ssl.keyStore=bin/amc/ActionManager/testadmin.jks	Truststore files are now local to Solution Directory.

The table below lists which properties have been added in Tivoli Directory Integrator 7.1.1:

Table 10. New properties in AM

New property (default)	Remarks
smtp.host= smtp.port= smtp.user= {protect}-smtp.password=	SMTP server details, added in TDI 7.0.
javax.net.ssl.trustStore=TDI_Install_dir/serverapi/testadmin.jks {protect}-javax.net.ssl.trustStorePassword=administrator javax.net.ssl.trustStoreType=jks javax.net.ssl.keyStore=TDI_Install_dir/serverapi/testadmin.jks {protect}-javax.net.ssl.keyStorePassword=administrator javax.net.ssl.keyStoreType=jks	Action Manager SSL Properties, added in TDI 7.1.
com.ibm.di.amc.am.encryption.keystore = TDI_Install_dir/testserver com.ibm.di.amc.am.encryption.key.alias = server com.ibm.di.amc.am.encryption.keystoretype = jks com.ibm.di.amc.am.encryption.transformation = RSA com.ibm.di.amc.am.stash.file = TDI_Install_dir/idisrv.sth	Action Manager encryption properties, added in TDI 7.1. These properties are similar to the encryption properties used by the server. For convenience the location of the stash file has been added as a property: com.ibm.di.amc.am.stash.file. By default the AM will reuse the server's keystore and stash file encryption/decryption of AM protected properties.

Configurations:

Certain Directory Integrator components/features have been modified or removed. Configurations that reference these need to be migrated manually. Here is a list of affected components/features:

- Checkpoint/restart functionality:

This functionality is removed in 7.0. This leaves Connectors that support Iterator mode with only the default ability to do a simple reconnect and automatically skip forward as many times as the number of successful reads. The assumption is that skipping forward this number of entries would get you

back to where you last left off. Most Tivoli Directory Integrator Connectors will not automatically attempt to do this, because the behavior can be indeterminate or not appropriate. However, the default behavior is specific per Connector. The ability to automatically skip forward as many times as the number of successful reads is a new reconnect option available to each Connector and is configured in the Connection Errors panel, see **The Configuration Editor -> The Connector Editor -> Connection Errors** in the *IBM Tivoli Directory Integrator V7.1.1 Users Guide*. If you require more than the ability to automatically skip entries processed, you need to use one of the following options in your solutions:

- Configure Delta for an Iterator mode for dynamically changing result sets.
 - Override the `on_connection_failure` hook and do custom reconnect logic.
- Derby/Cloudscape in embedded mode as System Store used by multiple JVMs:
Default and recommended behavior in Tivoli Directory Integrator 7.1.1 is running Derby in networked mode. If you continue to use Derby in embedded mode, considerations regarding multiple JVMs attempting to use the same database simultaneously still apply; see "Using Derby to hold your System Store" on page 177. For migrating databases, see "Migrating Cloudscape database to Derby" on page 86.
 - Exchange Changelog Connector:
This Connector is removed in v7.0. You may consider using the unsupported Exchange Changelog Connector that is now provided as an "example" in `TDI_install_dir/examples/ExchangeChangelogConnector`.
 - Btree Connector:
This Connector is removed from the default installation in v7.0. Use the System Store Connector instead as described in section "Migrating BTree tables and BTree Connector to System Store" on page 85; alternatively, use the (unsupported) Btree connector that is now provided as an "example" in `TDI_install_dir/examples/BTreeDBConnector`.
 - Domino Change Detection Connector:
This applies to 6.0 and 6.1 only.
The **Delivery Mode** parameter is removed and **State Key Persistence** will be used instead. The behavior of old configurations which use this parameter will be as follows:
 - If the **Delivery Mode** parameter is set to "Assured once and only once delivery" mode then the **State Key Persistence** parameter will be set to "After read" which is the same behavior – the synchronization state is saved right after the notes document is read.
 - If the **Delivery Mode** parameter is set to "Normal assured delivery" mode then a check for a valid **State Key Persistence** parameter is made. If such is not found then the value of the **State Key Persistence** parameter is set to "After read". If the parameter is found in the configuration then the original value of it is used.
 - IDS Changelog Connector:
The CRAM-MD5 option is no longer available in 7.0; you must manually choose another authentication mechanism.

In version 6.2 of Tivoli Directory Server the BEREncoder and BERDecoder classes have been moved from the `com.ibm.asn1` package to the `com.ibm.ldap.bp.asn1` package. Starting from Tivoli Directory Integrator v7.0 custom user solutions that directly use the old classes (`com.ibm.asn1.BEREncoder` and `com.ibm.asn1.BERDecoder`) need be updated to reflect this change.

- EMF XMLToSDO and EMF SDOToXML Function Components:
These are deprecated in 7.0. Consider other functionality in the future.
- DSMLv2 Parser:
This applies only to 6.0.
The "dsml.request" and "dsml.response" attributes have been removed. These attributes used to provide the raw request and response objects from the ITIM DSMLv2 library. If you have old configurations using any of these Attributes, you to edit your old configurations so that these Attributes are no longer used. All the data available through the raw request and response objects is also available through the other Attributes delivered by the DSMLv2 Parser.

- ITIM Agent Connector:

If you have used the ITIM Agent Connector in a previous version of Tivoli Directory Integrator, you may have to change the way you configure SSL connections. The ITIM Agent Connector in IBM Tivoli Directory Integrator 7.1.1 uses JSSE (Java based keystore or truststore) 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:

1. Import the ITIM Agent's certificate that was previously mentioned in the "CA Certificate File" parameter into the Tivoli Directory Integrator truststore 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`.

2. Configure this truststore in the "server authentication" section of the `global.properties` or `solution.properties` file:

```
## server authentication  
  
javax.net.ssl.trustStore=serverapi\tim.jks  
{protect}-javax.net.ssl.trustStorePassword=administrator  
javax.net.ssl.trustStoreType=jks
```

Now, the ITIM Agent Connector uses the same JSSE-based secure communications architecture as the rest of Tivoli Directory Integrator.

If you already have a truststore file configured in `global.properties` or `solution.properties`, then import the certificate into that store instead of creating a new one.

- XML Parser:

The pre-v7.0 XML Parser has been renamed and is now called the Simple XML Parser; the current XML Parser is a new parser with more functionality, especially regarding hierarchical objects. Config files created under earlier versions of Tivoli Directory Integrator referring to the XML Parser will, when imported into v7.1 and later, refer to the Simple XML Parser (as the class name has not changed). If you want to use the new XML Parser instead, you will need to change that in your AssemblyLines and/or Connectors. In order to have the new XML Parser behave like the old one did you must set the both Entry Tag and Value Tag parameters to the values used in the Simple XML Parser.

Note: The Simple XML Parser exports a script variable named "xmldom", which is not exported by the new XML Parser. The new XML Parser represents the deeper hierarchy with the Entry itself. Any logic that relies on the "xmldom" variable and cannot be reworked to make use of the hierarchical structure provided by the Entry class, must not migrate to the new XML Parser.

- Castor Java to XML and XML to Java Function Components:

From Tivoli Directory Integrator v7.1 onwards, the location of the Castor mapping file has changed, from `TDI_install_dir/jars/functions/di_castor_mapping.xml` to `TDI_install_dir/etc/di_castor_mapping.xml`.

Consequently, the default value for the **Castor Mapping File** parameter now reflects the new location.

- HTTP Client Connector:

From Tivoli Directory Integrator v7.1 onwards, the HTTP Client Connector has been modified to automatically send an HTTP "Connection" header with value "close" when it does not intend to reuse the TCP connection for more HTTP requests. The reason for this modification is to comply with HTTP 1.1 recommendation (<http://tools.ietf.org/html/rfc2616#section-14.10>).

This behavior is mandatory according to the HTTP 1.1 spec and previously you needed to code this yourself in the AssemblyLine.

- HTTP Server Connector:

From Tivoli Directory Integrator v7.1 onwards, HTTP Server Connector has been modified to use persistent HTTP connections by default. This means that one TCP connection can be used by the same

HTTP client for multiple HTTP requests (<http://tools.ietf.org/html/rfc2616#section-8.1>). HTTP clients may still send a "Connection" header with value "Keep-Alive", but it is no longer required in order to use a persistent connection. Idle TCP connections will be closed automatically after 20 seconds of inactivity.

Customized scripts:

If you have customized any of the Directory Integrator scripts (for example, adding items to the PATH or the LD_LIBRARY_PATH environment variables in the startup scripts - ibmdisrv, ibmditk), you should apply these customizations to the corresponding scripts of the new version.

Previous versions of Tivoli Directory Integrator used the (MY)CLASSPATH variable in these 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 do not have to do anything with the CLASSPATH variable; just make sure your library is in the correct place (typically in the jars/ directory) so it is found by Tivoli Directory Integrator. Alternatively, use the com.ibm.di.loader.userjars property in global.properties to point to your own directory to be included in the loader path. In Tivoli Directory Integrator 7.1.1, the property may specify several directories or jar files, separated by the Java Property "path.separator", which is ":" on Linux and ";" on Windows. The TDILoader for jar files searches directories recursively for files that contain classes and resources. Only files with a ".zip" or ".jar" extension are searched.

Added or replaced JAR files in the installation:

If you have added JAR files to the installation, you should copy them to the new version too.

IBM Tivoli Directory Integrator now requires and includes a Java 6 compliant JVM (J2SE version 1.6 SR9). 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 have to recompile and re-link your code.

If you have overwritten any of the original JAR files of the installation (for example, putting any required MQ jars in *TDI_install_dir/jars/3rdparty/IBM*), you should do the same with the new version.

A 64-bit Java Runtime Environment (JRE) is used now on Windows x86-64, Linux x86-64 and Linux s390. Compared to a 32-bit JRE, some performance degradation has been observed in some scenarios; you can still use the Windows x86-32 or Linux x86-32 installer for non-password plug-in activities if you believe you will have potential issues with performance degradation.

If you do use the 64-bit JRE, you need to be aware that 64-bit shared libraries will be needed for any custom component (connector, parser, FC) that depends on JNI.

Password Synchronizer configurations:

- Windows Password Synchronizer

Follow the steps described in "Migration from previous installations" in the Windows Password Synchronizer section of the *IBM Tivoli Directory Integrator V7.1.1 Password Synchronization Plug-ins Guide*.

- Other Password Synchronizers

There are no specific migration steps. Uninstall the old version, install v7.1.1 and configure it to suit your needs.

Backing up important data

It makes sense to backup each of the files of the original installation that you have modified in some way: property files, keystores, scripts, jars, and so forth.

Generally it is up to you to decide which items are important and have to be backed up. Here is an overview:

Files backed up by the Installer

Upgrade from version 6.0 to 7.1: If the Server feature is being upgraded the following files will be backed up:

```
TDI_install_dir\global.properties to TDI_install_dir\etc\global.properties.v60  
TDI_install_dir\serverapi\testadmin.jks to TDI_install_dir\serverapi\testadmin.jks.v60  
TDI_install_dir\serverapi\testadmin.der to TDI_install_dir\serverapi\testadmin.der.v60  
TDI_install_dir\serverapi\registry.enc to TDI_install_dir\serverapi\registry.enc.v60  
TDI_install_dir\serverapi\registry.txt to TDI_install_dir\serverapi\registry.txt.v60  
TDI_install_dir\idisrv.sth to TDI_install_dir\idisrv.sth.v60  
TDI_install_dir\testserver.jks to TDI_install_dir\testserver.jks.v60  
TDI_install_dir\testserver.der to TDI_install_dir\testserver.der.v60
```

In addition, configuration files and `solution.properties` will be backed up.

Upgrade from version 6.1.x to 7.1: If the Server feature is being migrated, the following files will be backed up: (The new suffix will be .v61 or .v611 depending on the previous version.)

```
TDI_install_dir\etc\global.properties to TDI_install_dir\etc\global.properties.v61x  
TDI_install_dir\serverapi\testadmin.jks to TDI_install_dir\serverapi\testadmin.jks.v61x  
TDI_install_dir\serverapi\testadmin.der to TDI_install_dir\serverapi\testadmin.der.v61x  
TDI_install_dir\serverapi\registry.enc to TDI_install_dir\serverapi\registry.enc.v61x  
TDI_install_dir\serverapi\registry.txt to TDI_install_dir\serverapi\registry.txt.v61x  
TDI_install_dir\idisrv.sth to TDI_install_dir\idisrv.sth.v61x  
TDI_install_dir\testserver.jks to TDI_install_dir\testserver.jks.v61x  
TDI_install_dir\testserver.der to TDI_install_dir\testserver.der.v61x  
TDI_install_dir\etc\reconnect.rules to TDI_install_dir\etc\reconnect.rules.v61x  
TDI_install_dir\etc\derby.properties to TDI_install_dir\etc\derby.properties.v61x  
TDI_install_dir\etc\jlog.properties to TDI_install_dir\etc\jlog.properties.v61x  
TDI_install_dir\etc\log4j.properties to TDI_install_dir\etc\log4j.properties.v61x  
TDI_install_dir\etc\tdisrvctl-log4j.properties to TDI_install_dir\etc\tdisrvctl-log4j.properties.v61x  
TDI_install_dir\etc\act-jlog.properties to TDI_install_dir\etc\act-jlog.properties.v611 (TDI 6.1.1 only)
```

In addition, configuration files and `solution.properties` will be backed up.

Upgrade from version 7.0 to 7.1: If the Server feature is being upgraded the following files will be backed up:

```
TDI_install_dir\etc\global.properties to TDI_install_dir\etc\global.properties.v70  
TDI_install_dir\serverapi\testadmin.jks to TDI_install_dir\serverapi\testadmin.jks.v70  
TDI_install_dir\serverapi\testadmin.der to TDI_install_dir\serverapi\testadmin.der.v70  
TDI_install_dir\serverapi\registry.enc to TDI_install_dir\serverapi\registry.enc.v70  
TDI_install_dir\serverapi\registry.txt to TDI_install_dir\serverapi\registry.txt.v70  
TDI_install_dir\idisrv.sth to TDI_install_dir\idisrv.sth.v70  
TDI_install_dir\testserver.jks to TDI_install_dir\testserver.jks.v70  
TDI_install_dir\testserver.der to TDI_install_dir\testserver.der.v70  
TDI_install_dir\etc\reconnect.rules to TDI_install_dir\etc\reconnect.rules.v70  
TDI_install_dir\etc\derby.properties to TDI_install_dir\etc\derby.properties.v70  
TDI_install_dir\etc\jlog.properties to TDI_install_dir\etc\jlog.properties.v70  
TDI_install_dir\etc\log4j.properties to TDI_install_dir\etc\log4j.properties.v70  
TDI_install_dir\etc\tdisrvctl-log4j.properties to TDI_install_dir\etc\tdisrvctl-log4j.properties.v70  
TDI_install_dir\etc\act-jlog.properties to TDI_install_dir\etc\act-jlog.properties.v70
```

In addition, configuration files, the workspace and solution.properties will be backed up.

Upgrade from version 7.1 to 7.1.1:

TDI_install_dir\etc\global.properties to *TDI_install_dir\backup_tdi\global.properties*
TDI_install_dir\serverapi\testadmin.jks to *TDI_install_dir\backup_tdi\testadmin.jks*
TDI_install_dir\serverapi\testadmin.der to *TDI_install_dir\backup_tdi\testadmin.der*
TDI_install_dir\serverapi\registry.enc to *TDI_install_dir\backup_tdi\registry.enc*
TDI_install_dir\serverapi\registry.txt to *TDI_install_dir\backup_tdi\registry.txt*
TDI_install_dir\idisrv.sth to *TDI_install_dir\backup_tdi\idisrv.sth*
TDI_install_dir\testserver.jks to *TDI_install_dir\backup_tdi\testserver.jks*
TDI_install_dir\testserver.der to *TDI_install_dir\backup_tdi\testserver.der*
TDI_install_dir\etc\reconnect.rules to *TDI_install_dir\backup_tdi\reconnect.rules*
TDI_install_dir\etc\derby.properties to *TDI_install_dir\backup_tdi\derby.properties*
TDI_install_dir\etc\jlog.properties to *TDI_install_dir\backup_tdi\jlog.properties*
TDI_install_dir\etc\log4j.properties to *TDI_install_dir\backup_tdi\log4j.properties*
TDI_install_dir\etc\tdisrvctl-log4j.properties to *TDI_install_dir\backup_tdi\tdisrvctl-log4j.properties*
TDI_install_dir\etc\tdimigglb-log4j.properties to *TDI_install_dir\backup_tdi\tdimigglb-log4j.properties*
TDI_install_dir\etc\updateinstaller-log4j.properties to *TDI_install_dir\backup_tdi\updateinstaller-log4j.properties*
TDI_install_dir\etc\it_registry.properties to *TDI_install_dir\backup_tdi\it_registry.properties*
TDI_install_dir\etc\tp.xml to *TDI_install_dir\backup_tdi\tp.xml*

In addition, configuration files from *TDI_install_dir\configs* folder, the workspace and solution.properties will be backed up.

Backup tools

These tools are used for backup/restore by the installer. They can also be used for manual migration.

backupamc/restoreamc

Use to backup/restore AMC configuration files.

backupamcdb/restoreamcdb

Use to backup/restore the AMC database.

backupam/restoream

Use to backup/restore the Action Manager (AM) database.

See “AMC and AM Command line utilities” on page 251 for more details.

Manual backup

Manual backup means copy the file to some dedicated backup folder. Conversely, restore means copy the file from the dedicated backup folder to its original location.

Note that in some cases you have to consider dependencies between files. You need to backup a group of interdependent files as a whole. Such groups of files are:

Derby database files

To backup a database, backup the whole folder that contains the database files. For example copy the *TDI_install_dir/TDISysStore* folder to backup the default System Store database or copy the *TDI_install_dir/bin/amc/tdiamcdb* folder to backup the default AMC database.

MQe queue manager files

Backup the whole folder of the MQe queue manager. For example copy the *TDI_install_dir/MQePWStore* folder to backup the default System Queue.

CE workspace files

Backup the whole workspace folder.

Migrating AMC 7.x configuration settings to another AMC deployment

The section explains the steps that are to be followed for migrating AMC configuration data to a new AMC deployment. These instructions are useful when migrating AMC 7.0 and later from ISC SE to Tivoli Integrated Portal (ISC embedded), Tivoli Integrated Portal (ISC embedded) to ISC SE, or to create a mirror instance of AMC on another machine.

1. Backup all of the AMC configuration files and data:
 - a. Stop the AMC that you are migrating configuration data from using the `stop_tdiamc.bat(sh)` script. This script stops the server on which AMC is deployed.
 - b. Execute the `backupamc.bat (.sh)` script specifying the directory in which the AMC configuration files need to be backed up.
2. Migrate all of the AMC configuration data to the new AMC instance:
 - a. If the ISC you are migrating the AMC configuration to, is residing on another machine you will need to get the AMC backup directory copied to the new machine.
 - b. Ensure the AMC being migrated to is stopped. See step 1a for information on stopping AMC.
 - c. Execute the `restoreamc.bat (.sh)` script specifying the directory you have the AMC configuration information backed up to. This command will place the AMC configuration files at the correct location in the AMC you are migrating to. This completes the migration process.

Notes:

1. The instructions are for migrating AMC 7.0 and later to any other ISC deployment.
2. The instructions assume that you already have AMC deployed already in both the system being migrated from and the system being migrated to using the TDI installer. The systems can be either running AMC in ISC SE or Tivoli Integrated Portal (ISC embedded).
3. If you are trying to migrate the AMC configuration to another AMC deployment on the same machine and you want all of the AMC commands shipped with your TDI deployment to use that AMC from that point on, you will need to update the configured ISC location in the AMC command line utility. This can be done using the following command: `setISCHome.bat(sh)`. The command takes as a parameter the location of the ISC installation directory, which is the installation location of Websphere for Tivoli Integrated Portal (ISC embedded) and the location of the embedded Web platform for ISC SE. The command needs to be executed between steps 1 and 2 mentioned above.

Converting from EventHandlers to corresponding AssemblyLines

EventHandlers do not exist in Tivoli Directory Integrator 7.1.1. Therefore, in order to replace the deleted functionality in old solutions, you need to migrate your EventHandler configurations to Server/Iterator or Changelog Connector configurations.

For each EventHandler a corresponding AssemblyLine must be created. Then a Server/Iterator Connector corresponding to the EventHandler must be inserted into the AssemblyLine "Feeds" section. Then the Connector parameters must be set – this is specific for each EventHandler/Connector pair, but generally the Connector parameters must be set to the same values as the corresponding EventHandler parameters (which usually have the same names).

Any processing configured in the EventHandler must be re-implemented in the AssemblyLine "Flow" section.

The functionality of the "enabled" EventHandler parameter (otherwise known as "Auto-start service") is also available for AssemblyLines. If you want your AssemblyLine to be started right after the Tivoli Directory Integrator Server is started, go to the **Solution Logging and Settings** section in the Navigator in your workspace in the Config Editor and add your AssemblyLine.

In general an EventHandler executes some piece of logic when a certain event occurs. "Event" has a different meaning for each EventHandler. For the HTTP EventHandler an "event" is an HTTP request. For the IBM Directory Server EventHandler an "event" is a change notification that comes from an IBM LDAP Directory.

Below are some general guidelines on migrating certain parts of a typical EventHandler. They are divided based on the titles of the UI tabs for an EventHandler in the pre-7.0 Config Editor:

Hooks

The "Prolog" hook of an EventHandler corresponds to the "Prolog – After Init" hook of an AssemblyLine. This hook is invoked for each incoming "event".

The "Epilog" hook of an EventHandler corresponds to the "Epilog – After Close" AssemblyLine hook. This hook is invoked once after each incoming "event" is processed.

In both the "Prolog" and "Epilog" EventHandler hooks the "event" Entry is accessible under the names "conn" and "event". However in the AssemblyLine hooks you should modify your script to use "work" instead of "conn" or "event".

The "Shutdown Request" hook of an EventHandler corresponds the "Shutdown Request" AssemblyLine hook.

Action Map

The Action Map of an EventHandler defines what actions should be taken when an "event" arrives. You should build the same actions into the logic of the AssemblyLine that you are preparing as a replacement of the EventHandler.

For example if the Action Map prescribed that a custom script should be executed if Attribute "x" of the event equals "3", then you could add an "IF" component to the AssemblyLine that checks for Attribute "x" being equal to 3 and executes a Script Component.

Logging

If you have configured custom log appenders for the EventHandler, you should configure the same appenders in the logging settings of the AssemblyLine(s) that you are preparing as a replacement for the EventHandler.

Config

These configuration parameters are specific to each EventHandler. See the subsections below for instructions on how to migrate them. The subsections are named after the corresponding Connectors.

TCP Server Connector

You must do the following to reproduce an old EventHandler's configuration into a new Connector's configuration:

1. Create a new AssemblyLine and insert the TCP Server Connector in it.
2. Set the **tcp.port** and **debug** Connector parameters to the values of the corresponding EventHandler parameters.
3. Set the **useSSL** and **requireClientAuth** Connector parameters to false (unchecked in the Config Editor).

Mailbox Connector

There is no need to migrate existing configurations that use the Tivoli Directory Integrator v6.0 Mailbox Connector, because the Tivoli Directory Integrator v7.1.1 Mailbox Connector is compatible with the Tivoli Directory Integrator v6.0 Mailbox Connector.

Configurations that use the Mailbox EventHandler, however, need to be migrated by following these steps:

1. Create a new AssemblyLine and insert the Mailbox Connector in it.
2. Copy the contents of the **mailServer** EventHandler parameter to the Connector parameter with the same name.
3. Set the **mailProtocol** Connector parameter to the value of the EventHandler parameter with the same name.
4. Copy the contents of the **mailUser** and **mailPassword** EventHandler parameters to the Mailbox Connector parameters with the same names.
5. Copy the contents of the **mailFolder** EventHandler parameter to the Connector parameter with the same name.
6. Copy the contents of the **pollInterval** EventHandler parameter to the Connector parameter with the same name.
7. If the enabled EventHandler parameter is true, add your AssemblyLine to the "Config -> AutoStart" folder in the Config Editor; thus the Tivoli Directory Integrator server will start your AssemblyLine on startup.
8. If the **debug** EventHandler parameter is true, set the Connector parameter with the same name to true.

JMX Connector

Existing configurations that use the Tivoli Directory Integrator v6.0 JMX EventHandler can be transformed into Tivoli Directory Integrator v7.1.1 configurations that use the JMX Connector in the following way:

1. Create a new AssemblyLine and insert the JMX Connector in it.
2. Copy the contents of the **eventTypes** JMX EventHandler parameter to the JMX Connector parameter with the same name.
3. Select "local" for the **mode** Connector parameter.
4. Leave the **url** Connector parameter blank.
5. Set the **allMBeans** Connector parameter to true.
6. Leave the **mBeanTypes** Connector parameter blank.

SNMP Server Connector

The Tivoli Directory Integrator 7.1.1 SNMP Server Connector provides all features of the Tivoli Directory Integrator v6.0 SNMP EventHandler except support for single-threaded mode. The Tivoli Directory Integrator 7.1.1 SNMP Server Connector works in multi-threaded mode only. If you need to migrate an existing Tivoli Directory Integrator v6.0 configuration using the SNMP EventHandler to a Tivoli Directory Integrator v7.0 configuration, which uses an AssemblyLine with the SNMP Server Connector, you need to do the following:

1. Create a new AssemblyLine.
2. Insert into the AssemblyLine an instance of the SNMP Server Connector.
3. Set the **udp.port** Connector parameter to the value this parameter has in your SNMP EventHandler configuration.
4. Set the **snmp.community** Connector parameter to the value this parameter has in your SNMP EventHandler configuration.
5. If your SNMP EventHandler used to be configured to be "Auto-started" by the TDI Server, add your new AssemblyLine to the "Config -> AutoStart" folder of the Config Editor.

IBM Directory Server Changelog Connector

Existing configuration that use the IBM Directory Server EventHandler can be migrated to use the IBM Directory Server Changelog Connector as follows:

1. Set the following Connector parameters to the values of the EventHandler parameters with the same names: **ldapUrl**, **ldapUsername**, **ldapPassword**, **ldapAuthenticationMethod**, **ldapUseSSL**, **ldapSearchBase**.
2. Leave the **jndiExtraProviderParams** Connector parameter empty.
3. Set the **iteratorStateKey** Connector parameter to some unique identifier, one that has no corresponding state saved in the System Store.
4. Set the **nsChangenumber** Connector parameter to the next change number that the EventHandler would process. The last change number that the EventHandler has processed is normally stored in an external properties file, referenced by its **ldapChangeNumberFileName** parameter.
5. Set the **stateKeyPersistence** Connector parameter to "After read" (the EventHandler writes the last received change number to its file backend after it reads a changelog entry and before it dispatches it for processing).
6. Set the **mergeMode** Connector parameter to "Merge changelog and changed data". This will ensure that the changelog attributes (changenumber, targetdn, ...) appear as attributes of the Entry.
7. Set the **useNotifications** Connector parameter to true.
8. Set the **batchRetrieval** Connector parameter to false.

Note: As opposed to the EventHandler, the Connector does not let you select a part of the directory tree, for whose notifications it will listen – it subscribes for changes in the whole directory tree (the Connector does not have equivalents of the **ldapEventBase** and **ldapSearchScope** EventHandler parameters). If this is critical for you, you can implement some custom filtering in your solution to overcome this limitation of the Connector.

HTTP Server Connector

A configuration that uses the HTTP EventHandler can be migrated to use the HTTP Server Connector like this:

1. Set the **tcpPort** Connector parameter to the value of the **Port** parameter of the EventHandler.
2. Leave the **backlog** Connector parameter empty.
3. Set the **contentType** Connector parameter to "text/html".
4. Set the **tcpDataAsProperties** Connector parameter to true (the EventHandler always returns the TCP information as properties).
5. Set the **headersAsProperties** Connector parameter to the value of the **headersAsProperties** of the EventHandler.
6. Set the **httpAuth** Connector parameter to true, if the EventHandler uses HTTP basic authentication (that is if it has a configured authentication Connector).
7. If the EventHandler uses HTTP basic authentication, set the **authRealm** Connector parameter to the value of the **authrealm** EventHandler parameter. If the **authrealm** EventHandler parameter is missing or empty, set the **authRealm** Connector parameter to "IBM-Directory-Integrator".
8. Set the **authConnector** Connector parameter to the value of the **AuthConnector** parameter of the EventHandler.
9. Set the **useSSL** Connector parameter to the value of the **useSSL** parameter of the EventHandler.
10. Set the **needClientAuth** Connector to false (the EventHandler does not support SSL client authentication).
11. Set the **msgChunked** Connector parameter to false (the EventHandler does not support chunking of HTTP responses).

LDAP Server Connector

A configuration that uses the LDAP Server EventHandler can be migrated to use the LDAP Server Connector as follows:

1. Set the **ldapPort** Connector parameter to the value of the **tcp.port** parameter of the EventHandler.

2. Leave the **backlog** Connector parameter empty.
3. Set the **ldapUseSSL** Connector parameter to the value of the **ldapUseSSL** parameter of the EventHandler.
4. Set the **charset** Connector parameter to the value of the **charset** parameter of the EventHandler.
5. Set the **ldapBinaryAttributes** Connector parameter to the value of the **binary** parameter of the EventHandler.

Sun Directory Change Detection Connector

The LDAP EventHandler catches notifications about changes in a directory tree. The EventHandler does not use a changelog, so it receives only real-time notifications. The Sun Directory Change Detection Connector offers basically the same functionality when run in real-time delivery mode. There are a few differences though:

The Connector does not have equivalents for the **ldapSearchFilter** and **ldapSearchScope** EventHandler parameters. To achieve the same functionality as in the EventHandler, you should implement some custom filtering that limits the set of received notifications.

The schema of the returned data differs between the Connector and the EventHandler. The Connector applies delta tagging to each Entry it returns, while the EventHandler provides the type of the change in the "ldap.operation" property. For details on the schema consult the documentation of each component.

Once the considerations above are resolved, you can migrate an existing configuration with the LDAP EventHandler to use the Sun Directory Change Detection Connector like this:

1. Set the following Connector parameters to the values of the EventHandler parameters with the same names: **ldapUrl**, **ldapUsername**, **ldapPassword**, **ldapAuthenticationMethod**, **ldapUseSSL**, **ldapSearchBase**.
2. Leave the **jndiExtraProviderParams** Connector parameter empty.
3. Set the **deliveryMode** Connector parameter to "Realtime" (the EventHandler does not use a changelog, it only catches real-time notifications).
4. Set the **mergeMode** Connector parameter to "Return only changed data" (no changelog is used in real-time delivery mode by the Connector).

Active Directory Change Detection Connector

The migration from the AD Changelog EventHandler to the Active Directory Change Detection Connector is straight forward in the most aspects since the EventHandler itself has incorporated the older version this connector – Active Directory Changelog Connector in order to obtain changes from the AD.

Similar to the EventHandler the corresponding Connector can also be interrupted any time during the synchronization process, in that case it will store its state in the User Property Store. Both the EventHandler and the Connector rely on the uSNChanged mechanism in this process, by storing the USN number in the property store. They also offer an API for retrieving the current USN synchronization values. The difference is that the EventHandler `getUSNvalues` method returns an Entry with Attributes:

```
START_USN
END_USN
CURRENT_USN_CREATED
CURRENT_USN_CHANGEDT
```

whereas the Connector returns the current synchronization value as *long*.

Another difference is that the AD EventHandler initializes internally an LDAP Connector in order to block and receive change notifications. This behavior can also be simulated in the ADCD Connector by enabling the **useNotifications** parameter.

The following steps should be performed in order to migrate from an EventHandler-based solution to Connector-based one:

1. Create a new AssemblyLine with an instance of an Active Directory Change Detection Connector in Iterator mode.
2. Set the **ldapUrl**, **ldapUsername**, **ldapPassword** and **ldapAuthenticationMethod** to the values these connection parameters have in the EventHandler configuration.
3. Specify whether SSL connection is used according to the value in the old configuration.
4. Copy the content of the **ldapSearchBase** EH parameter to the same in the Connector configuration
5. Copy the content of the **persistentParameterName** EH parameter to the **persistentStateKey** Connector parameter.
6. Set the parameter **useNotifications** to true.
7. Set the **startAt** parameter according to the value in EH.
8. Leave the other Connector parameters as they are.
9. Transfer any logic in the Action Map section of the EventHandler to be invoked from the new AL.

z/OS LDAP Changelog Connector

The migration from the z/OS LDAP Changelog EventHandler to the z/OS LDAP Changelog Connector is facilitated by the fact that this changelog connector is explicitly developed to replace the EventHandler. Both the EventHandler and the Connector rely on a poll mechanism for extracting the changelog information since they do not support the Unsolicited Event Notification.

The main difference consists in storing the state key. The EventHandler uses a plain file passed as parameter - **ldapChangeNumberFileName** in its configuration to keep track of the last used changenumber, whereas the Connector takes advantage of the "Iterator State Key" mechanism in Tivoli Directory Integrator. Therefore, in order to migrate a solution from the from the z/OS LDAP Changelog EventHandler to the z/OS LDAP Changelog Connector, the value of the changenumber in the **ldapChangeNumberFileName**-specified file must be passed to the Connector for example with script before the initial start of the iteration, like this script snippet:

```
Com.ibm.di.store.StoreFactory.getDefaultPropertyStore().setProperty("changenumber",
    new Long(changeNumber));
```

where **changenumber** is the name of the Iterator State Key and *changeNumber* is the value obtained from the store file used by the EventHandler.

An alternative to this solution is to pass the *changeNumber* value to the **nsChangenumber** parameter in the configuration of the Connector.

The following steps should be performed in order to migrate from an EH-based solution to Connector-based one:

1. Create a new AssemblyLine with an instance of the z/OS LDAP Changelog Connector in Iterator mode.
2. Set the **ldapUrl**, **ldapUsername**, **ldapPassword** and **ldapAuthenticationMethod** to the values these connection parameters have in the EventHandler configuration.
3. Specify whether SSL connection is used according to the value in the old configuration.
4. Copy the content of the **ldapSearchBase** EH parameter to the same in the Connector configuration.
5. Apply one of the solutions described above to supply the Iterator State Key with the last "changenumber" value used for synchronization by the EH.
6. Copy the content of the **pollInterval** EH parameter to the **nsSleepInterval** Connector parameter.
7. Leave the other Connector parameters as they are.
8. Transfer any logic in the Action Map section of the EventHandler to be invoked from the new AL.

DSMLv2SOAPServerConnector

The migration from the DSMLv2 EventHandler to the DSML v2 SOAP Server Connector requires rework of the AssemblyLines that are previously used with the EventHandler, so that they can be integrated in the solution with the DSMLv2 SOAP Server Connector. This is because the core architecture has changed; now a single AssemblyLine processes all operations. Therefore, all the old AssemblyLines logic responsible for handling the different types of DSMLv2 operations should be incorporated into the new AssemblyLine containing the DSMLv2 Soap Server Connector, or should be invoked using a AssemblyLine Connector. For this purpose branching components can be used in order to separate the logic for the specific DSMLv2 operations (available in the `dsml.operation` Attribute).

The migration of a configuration with the DSMLv2 EventHandler to a similar one with the DSMLv2 SOAP Server Connector consists of the following steps:

1. Create a new AssemblyLine with an instance of a DSMLv2 SOAP Server Connector in Server mode.
2. Copy the content of the EH `port` parameter to the `dsmlPort` Connector parameter.
3. Set the `authRealm`, `useSSL`, `binaryAttributes` and `msgChunked` to the values these connection parameters have in the EventHandler configuration.
4. Create a branch component for each of the DSMLv2 operations listed as parameters in the EH configuration and apply in the branches the logic implemented in the corresponding old AssemblyLine, either by transferring there the appropriate AL components or by invoking the old AL itself using an AssemblyLine connector. In both cases the naming context will no longer be needed.
5. Copy the content of the twoEH `WaySSL` parameter to the `needClientAuth` Connector parameter.
6. The EH Attribute `headerAsProperties` cannot be passed to the Connector, since the HTTP parser it initializes internally is configured to always set this value to "false". Therefore, in case the solution accesses headers as properties, it should be modified to use Attributes for this purpose (`getAttribute()` instead of `getProperty()`).
7. For compliance, the `soapbinding` Connector Attribute should be set to "false" since the DSMLv2 parser internally used by the EH does not take advantage of it.
8. In case an `authConnector` is specified in the configuration of the DSMLv2 EventHandler, then the HTTP basic authentication of the Connector must be enabled and the appropriate logic must be implemented in the "After Accepting connection" hook (for example, initialize the authenticator Connector and call its `lookup()` method using an Entry with Attributes "username" and "password" as search criteria. Similar to the EventHandler the authentication is to be considered successful in case an Entry is returned).
9. The `indentoutput` parameter of the DSMLv2 parser internally used by the Connector cannot be set in contrast to the one used by the EH.

Migrating BTree tables and BTree Connector to System Store

The BTree Connector is deprecated, and is now only provided as an unsupported example. Therefore, 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 System Store 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.

Changing a solution to use the System Store Connector instead of the BTree Connector for storing Tivoli Directory Integrator Entries is straight forward since both connectors follow the same logic when specifying Key Attribute Name and Selection Mode attributes. The only difference is that instead of the underlying BTree database the System Store Connector has to use predefined a database (for example the embedded Derby database) and specify a table to store into.

Storing other Java objects using the System Store Connector differs significantly from storing them with BTTree and will require more elaborate transformation. The following solution, which puts Java objects in the underlying BTTree database, cannot be directly applied to the System Store Connector, since it does not provide direct access to the backend database:

```
scripts var bt = system.getConnector("btreedb");
bt.initialize (null); var db = bt.getDatabase();
db.insert ("my key", new java.lang.String("my value"));
var value = db.search ("my key"); value = value + " - modified";
db.replace ("my key", value);
```

Instead of this the standard methods (`put()`, `find()` and `modify()`) from the Connector API can be used, but the object should be first wrapped into an `Entry` object, which subsequently can be stored in the System Store.

Migrating Cloudscape database to Derby

IBM Tivoli Directory Integrator 7.1.1 uses Derby v10.5.3 as its bundled database, used by default by the System Store. You will need to migrate your existing Cloudscape or Derby databases (created using previous versions of Tivoli Directory Integrator) to be able to use Tivoli Directory Integrator 7.1.1. Derby v10.5.3 drivers that are shipped with Tivoli Directory Integrator 7.1.1 cannot be used to communicate with older versions of Cloudscape.

For details, and information on differences between Cloudscape/Derby v10.5.3 and its prior versions, refer to the following web page:<http://publibfp.boulder.ibm.com/epubs/html/c1894710.html>.

Notable differences that have an immediate impact are as follows:

- The long varbinary data type is no longer supported. Instead, BLOB datatype has been introduced (making Derby compatible with DB2®). For this reason, all SQL Statements that made use of long varbinary datatype must now be modified to use BLOB.
- JDBC Java package names have changed from `com.ibm.db2j.*` in previous releases to `org.apache.derby.*` in Derby v10.
- The JDBC URL for Derby (embedded/network mode access) v10 is different from Cloudscape v5.1. Hence the JDBC properties mentioned in `global.properties` / `solution.properties` have also been modified for the current version of Tivoli Directory Integrator.

Table 11. JDBC URL differences

Connection type	Cloudscape v5.1	Derby v10
Embedded Derby / Cloudscape	<code>jdbc:db2j:</code>	<code>jdbc:derby:</code>
DB2 JDBC Universal Database Driver (Network mode)	<code>jdbc:db2j:net</code>	<code>jdbc:derby:net</code> (Not recommended to use)
DerbyClient Driver	-	<code>jdbc:derby</code> (Recommended)

Fortunately, the Derby team have provided a migration utility that migrates a Cloudscape v5.1 database to a new Derby v10 database. It migrates all the tables and their corresponding data into a newly generated Derby v10 database. It modifies all tables with varbinary datatype to BLOB datatype, hence making the migration process quite painless.

This utility is bundled with Tivoli Directory Integrator 7.1.1, in the `TDI_install_dir/tools/CSMigration` folder, along with a wrapper script that invokes the migration tool, called `migrateCS.bat(sh)`. To migrate a Cloudscape 5.1 System Store Database created using TDI v6.0 to Derby v10, you have to invoke the `migrate` script in the following manner:

```
migrateCS [Path_of_CloudscapeV51_Database] [Path_of_new_DerbyV10_Database]
```

You may need to give some thought to the location of the new Derby database. In Tivoli Directory Integrator v6.0 and v6.1.x, the System Store database often was located in the installation directory of Tivoli Directory Integrator; this is an unfortunate location for many reasons. For Tivoli Directory Integrator 7.1.1 we strongly recommend you use a Solution Directory, away from the installation directory.

Besides migration of data, you also need to modify your `global.properties` / `solution.properties` files (using the migration tool or manually) to incorporate the new JDBC URL parameters.

Migrating global and solution properties files using migration tool

Use the `tdimiggl` tool located in the `TDI_install_dir/bin` directory to migrate any `global.properties` file starting with Tivoli Directory Integrator 6.x to 7.1.1. The filename is `tdimiggl.bat` on Windows and `tdimiggl.sh` on UNIX/LINUX. Use the `tdimiggl-4log4j.properties` file to control logging for `tdimiggl.bat(sh)`.

The usage if the command is as follows:

```
tdimiggl -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
-s dir - Working directory where the solution directory is located.
-v - Enable verbose mode
-? - Prints the usage statement

During the installation of Tivoli Directory Integrator, the installer backs up the existing `global.properties` file; and then calls this command, in order to migrate the `global.properties`.

The migration tool tries to migrate a `global.properties` file (or `solution.properties` file if required) up to the latest Tivoli Directory Integrator version. The tool (`tdimiggl`) makes no assumptions about which release the `global.properties` files starts from and can handle `global.properties` files starting at Tivoli Directory Integrator version 6.0. The tool also tries to apply all migration changes unless a particular migration step is specifically declared inappropriate for migration by the migration tool. For these cases, perform the migration steps manually.

The activities of the migration tool are broken down into stages. In sequence, the tool:

1. Checks whether you have to migrate your Derby (Cloudscape) database (Tivoli Directory Integrator 6.0 migration).
2. Performs all of the migration actions in the following order:
 - a. Delete actions.
 - b. Add actions.
 - c. Derby (Cloudscape) migration file changes (only if necessary and only for Tivoli Directory Integrator 6.0 migrations).
 - d. Migration modify actions.
3. Calls the Derby (Cloudscape) migration tool `migrateCS` to migrate the database up to the current Derby version (only for Tivoli Directory Integrator 6.0 migrations).
For each action set (migration modify actions for example), the migration tool tries to perform the migration actions starting from the earliest release to the latest release. For migration from Tivoli Directory Integrator 6.0, the caller must separately invoke the Derby (Cloudscape) migration tool to migrate the database up to the current Derby version. The `tdimiggl` tool only makes the required Derby (Cloudscape) modifications to the properties file itself.
4. Uses log4j logging APIs for logging error messages.

The log4j configuration file is specified in the startup script (the bat or sh) file. The command uses a file called `tdimiggbl-log4j.properties` to set up the log4j logging. The command changes directory to the solution directory and therefore uses the `tdimiggbl-log4j.properties` file in the solution directory if the Tivoli Directory Integrator installation directory is not specified.

Migrating Password plug-ins properties files using migration tool

Tivoli Directory Integrator 7.1.1 contains a migration utility to upgrade the `pwsync.props` files for each of the installed Password plug-ins. The utility is called `migpwsync` and is provided to migrate the `pwsync.props` files read by both the native plug-in and the JavaProxy.

The `migpwsync` utility is shipped in the `TDI_install_dir/pwd_plugins/bin` directory.

The utility has the following options:

- `-?` – using this option the utility will print the help information and will exit.
- `-v` – using this option the utility will print more verbose information to the standard output
- `-f` – this is a required option used to provide the location of the `pwsync.props` file.
- `-b` – this is the option that specifies the location of the file that will be used as backup. This is an optional field and if not provided the value of the `-f` option will be used with ".backup" appended.
- `-n` – this option specifies the file location where the migrated information will be written to. This is an optional field and if not provided the value of `-f` will be used as the place to output the migrated configuration.

Examples:

- Migrating the configuration file of PAM plug-in :
`# TDI_install_dir/pwd_plugins/bin/migpwsync.sh -f TDI_install_dir/pwd_plugins/pam/pwsync.props`
- Migrating the configuration file of Windows plug-in :
`> TDI_install_dir\pwd_plugins\bin\migpwsync.bat - f TDI_install_dir\pwd_plugins\windows\pwsync.props`

Notes:

1. The installer will update all the `pwsync.props` files setup by it in the `TDI_install_dir/pwd_plugins` directory during installation. If you have moved any of the `pwsync.props` files then you need to be manually migrate it using a command similar to the ones above.
2. The `migpwsync` utility changes the current directory to the plug-ins home directory (`TDI_install_dir/pwd_plugins`.) The provided file paths will be considered relative to that directory, if they are not absolute paths.
3. After migrating old `pwsync.props` file, add the following ActiveMQ related properties, if you want to configure ActiveMQ as default the JMS Password store:
 - `jmsDriverClass=com.ibm.di.plugin.pwstore.jms.driver.ActiveMQ`
 - `jms.broker=<JMS Server address>`. For example, `jms.broker=tcp://<activeMQhost>:61616` or `jms.broker=ssl://<activeMQhost>:61617`
4. To configure ActiveMQ as the default JMS Password store, set the `jms.clientId` property in the `pwsync.props` file.

Chapter 6. Security and TDI

Introduction

Security features are found throughout IBM Tivoli Directory Integrator (Tivoli Directory Integrator). Some features secure access into remote systems from Tivoli Directory Integrator, others protect access into Tivoli Directory Integrator 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 Tivoli Directory Integrator in a stand-alone mode in a secured environment. However, the features come in handy when other systems must communicate with Tivoli Directory Integrator, such as through the remote Web Admin Console (AMC) management tool or the Tivoli Directory Integrator Remote Server API. Furthermore, if multiple people have access to the Tivoli Directory Integrator server it could be necessary to protect access to confidential data, as well as maintain the integrity of the integration rules that Tivoli Directory Integrator executes.

This chapter explains the following features:

1. "Manage keys, certificates and keystores"
2. "Secure Sockets Layer (SSL) Support" on page 94
3. "Remote Server API" on page 100
4. "Tivoli Directory Integrator Server Instance Security" on page 121
5. "Miscellaneous Config File features" on page 129
6. "Web Admin Console Security" on page 134
7. "Summary of configuration files and properties dealing with security" on page 131
8. "Miscellaneous security aspects" on page 134

This guide does not describe all the security capabilities of the individual Tivoli Directory Integrator components. Some common elements are described in "Miscellaneous security aspects" on page 134, however for individual elements of security configuration in the individual TDI components, consult the *IBM Tivoli Directory Integrator V7.1.1 Reference Guide*.

Manage keys, certificates and keystores

Background

The main uses of cryptographic keys in the product are SSL (see section "Secure Sockets Layer (SSL) Support" on page 94) and encryption (see section "Tivoli Directory Integrator Server Instance Security" on page 121).

For detailed information on security concepts and how they are used in the IBM JVM, see <http://www.ibm.com/developerworks/java/jdk/security/>.

Public/private keys and certificates

SSL and asymmetric encryption algorithms such as RSA (which is the default encryption algorithm of the Server) use public/private keys. Public and private keys have a one-to-one correspondence – matching public and private keys are called a "key pair".

Normally inside a keystore a public key comes wrapped in an X.509 certificate. Most keystore operations actually involve the whole public key certificate and not only the public key.

Again in most cases inside a keystore a private key is accompanied by the corresponding public key certificate.

Secret keys

Secret keys are used by symmetric encryption algorithms such as DES, AES and RC4. Note that some keystore formats such as JKS and PKCS#12 do not support secret keys.

You cannot use secret keys for SSL (the SSL protocol actually generates secret keys on the fly, but normally you don't have control over them).

Keystores

A keystore, as the name implies, provides storage for keys. It can be a file or a hardware device. The most popular keystore file formats used by Java programs are JKS, JCEKS and PKCS#12. See the following table for comparison:

Table 12. Keystore file formats

Keystore file format	Origin	Store public/private keys and certificates	Store secret keys
JKS	Proprietary	Yes	No
JCEKS	Proprietary	Yes	Yes
PKCS#12	Standard	Yes	No

Note that the only one of the above keystore formats that can store secret keys is JCEKS. Also in general JCEKS offers greater protection than JKS. JKS, JCEKS and PKCS#12 keystores are protected by a password. Furthermore, each private or secret key inside a keystore can be protected by an individual password. Public key certificates do not have passwords, because normally there is no need to keep them secret.

Keys for SSL

For detailed information on using SSL with the IBM JVM see: <http://www.ibm.com/developerworks/java/jdk/security/60/secguides/jsse2Docs/JSSE2RefGuide.html>.

To use SSL you need to provide a set of public/private keys. You cannot use secret keys for SSL.

An SSL connection has two sides – the SSL server side and the SSL client side. Each side has two keystores – an SSL keystore and an SSL truststore. Note that the word "keystore" is used both to mean a store of keys and an SSL keystore. So SSL keystore and SSL truststore are both keystores. In fact, the SSL keystore and the SSL truststore are only logical roles and it is perfectly legal to use the same physical keystore file for both. The SSL keystore contains a private key that is used to prove the authenticity of this SSL side to the other side of an SSL connection. The SSL truststore contains public key certificates of trusted parties.

1. To setup keys for your SSL server, you can: Generate a private key and a corresponding self-signed public key certificate and put it in your SSL keystore. (see section "Generate a public/private key pair and a self-signed certificate"). This step is needed only if your side of the SSL connection has to prove its authenticity to its peers – that is if you are the SSL server or if you are the SSL client and client authentication is required.
2. [Optional] Obtain a certificate from a Certificate Authority and replace your self-signed certificate with it. (see section "Import public key certificate in a keystore")
3. [Optional] Export the public key certificate of your private key and distribute it to the SSL parties that will interact with you. (see section "Export public key certificate from a keystore") If you are using a certificate from a Certificate Authority then it will be enough for others to have only the certificate of the Certificate Authority itself.
4. Import certificates of trusted parties in your SSL truststore (see section "Import public key certificate in a keystore"). This step is mandatory for SSL clients. For SSL servers it is necessary only if client authentication is required.

Note: If you are using the default properties to configure SSL (javax.net.ssl.*), the SSL keystore should contain exactly one private key, because there is no way to specify which key will be used.

Keys for encryption

For encryption you have two alternatives:

- use a public/private key pair
- use a secret key

For public key encryption the most popular algorithm is RSA. Note that other popular public key algorithms such as DiffieHellman (key exchange) and DSA (digital signature) cannot be used for encryption.

Generally encryption with secret keys is much faster and much more secure than encryption with public keys. However, by default the Directory Integrator Server uses public key encryption with RSA to preserve compatibility with earlier versions.

Tools

The IBM JVM provides two utilities for working with keys and certificates - keytool and Ikeyman. keytool is a command-line utility that is popular in the Java community. Ikeyman is a GUI tool from IBM, which provides many of the features of 'keytool'. Both tools are located in the *TDI_install_dir/jvm/jre/bin* folder. For detailed information about these tools see the documentation of the IBM JVM:
<http://www.ibm.com/developerworks/java/jdk/security/>

Default keystores shipped with the product:

Table 13. Tivoli Directory Integrator keystores

Keystore location	Keystore password	Trusted public keys	Private keys
<i>TDI_install_dir/testserver.jks</i>	server	admin	server
<i>TDI_install_dir/serverapi/testadmin.jks</i>	administrator	server	admin

List the contents of a keystore

Use the list command of keytool. For example the following command lists information about the keys (alias and type) inside the keystore file mystore.jck; the format of the keystore is JCEKS and its password is "mystorepass":

```
keytool -list -storetype jceks -keystore mystore.jck -storepass mystorepass
```

Create keys

Generate a public/private key pair and a self-signed certificate

For example the following keytool command generates an RSA public/private key pair with alias "myserverkey" and a X.509 self-signed public key certificate:

```
keytool -genkeypair -alias myserverkey -dname cn=myserver.mydomain.com  
-validity 365 -keyalg RSA -keysize 1024  
-keystore mystore.jceks -storetype jceks -storepass mystorepass  
mystorepass
```

The distinguished name of the owner of the certificate is "cn=myserver.mydomain.com", which should be the same as the DNS name of the server that will use the self-signed certificate for SSL (for public key encryption the content of the certificate does not matter much). The certificate is valid for 365 days. The size of the generated RSA key is 1024 bytes. The password of the private key is "mykeypass". The key pair is stored in a keystore file mystore.jck with format JCEKS (if the file does not exist, it will be created). The password of the keystore is "mystorepass".

The mystore.jck keystore can be used as an SSL keystore of a server program that runs on the "myserver.mydomain.com" host. The keystore also contains a public key certificate for the private key, so it can be used as an SSL truststore for clients that connect to the server on "myserver.mydomain.com". (Although to give your private key to clients is completely unnecessary and generally a bad security practice.)

Obtain a certificate from a Certificate Authority

Normally the process of acquiring and using CA-signed certificates goes like this:

First a key pair and a self-signed certificate is generated (see section "Generate a public/private key pair and a self-signed certificate"). 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, replacing the self-signed certificate.

For example using keytool you can generate a Certificate Signing Request for the "myserverkey" key from the mystore.jck keystore like this:

```
keytool -certreq -file myreq.csr -alias myserverkey -keypass mykeypass  
-storetype jceks  
-keystore mystore.jck -storepass mystorepass
```

This command creates a Certificate Signing Request in the myRequest.csr file for the public key with alias "myserverkey". The created Certificate Signing Request now can be sent to a Certification Authority. When the new certificate arrives, you can import it in the keystore as described in section "Import public key certificate in a keystore". The following keytool command generates a 256 bit AES key with alias "myseckeys":

```
keytool -genseckeys -keyalg AES -alias myseckeys -keysize 256 -keypass mykeypass  
-storetype jceks  
-keystore mystore.jck -storepass mystorepass
```

The new key is stored in a JCEKS keystore file mystore.jck with password "mystorepass". The password that protects the secret key is "mykeypass".

Copy key from one keystore to another

For example you can copy the key pair created in section "Generate a public/private key pair and a self-signed certificate" with the following keytool command:

```
keytool -importkeystore -srckeystore mystore.jck -destkeystore myotherstore.jks  
-srcstoretype jceks  
-deststoretype jks -srcstorepass mystorepass -deststorepass myotherstorepass  
-srcalias myserverkey  
-destalias myotherserverkey -srckeypass mykeypass -destkeypass myotherkeypass
```

The copy will be stored under alias "myotherserverkey" in the JKS keystore file myotherstore.jks (if it does not exist the file will be created).

Convert keystore from one format to another

For example you can convert the JCEKS keystore created in section "Generate a public/private key pair and a self-signed certificate" to a JKS keystore myotherstore.jks with the following keytool command:

```
keytool -importkeystore -srckeystore mystore.jck -destkeystore  
myotherstore.jks -srcstoretype jceks  
-deststoretype jks -srcstorepass mystorepass -deststorepass  
myotherstorepass
```

The command will eventually ask for the password of each individual private or secret key inside the source keystore. Note that JKS and PKCS#12 keystores cannot hold secret keys. You should not try to convert a keystore that contains secret keys to either JKS or PKCS#12.

Export public key certificate from a keystore

The following command exports the public key certificate created in section "Generate a public/private key pair and a self-signed certificate" to a binary file myserverkey.der:

```
keytool -exportcert -alias myserverkey -file myserverkey.der  
-storetype JCEKS -keystore mystore.jck  
-storepass mystorepass
```

The resulting .der file contains the DER encoding of the X.509 certificate. It is a binary file. To get the same binary data in text form (base-64 encoded form of the DER encoding of the X.509 certificate) use the "-rfc" option of keytool:

```
keytool -exportcert -alias myserverkey -file myserverkey.arm  
-storetype JCEKS -keystore mystore.jck  
-storepass mystorepass -rfc
```

Import public key certificate in a keystore

To import a new trusted certificate in a keystore use a command like this:

```
keytool -importcert -alias myserverkey -file myserverkey.der  
-storetype JCEKS -keystore mystore.jck  
-storepass mystorepass
```

keytool will attempt to verify the signer of the certificate which you are trying to import. This means constructing a certificate chain from the imported certificate to some other trusted certificate. If a chain cannot be established, keytool will ask you whether you are certain that the certificate needs to be imported.

To import a certificate that is a response from a Certificate Authority to a Certificate Signing Request (this means you already have a private key in the keystore for that certificate) use a command like this:

```
keytool -importcert -alias myserverkey -keystore mykeystore -file  
myserverkey.der -storetype JCEKS -keystore mystore.jck  
-storepass mystorepass
```

Note that when you import a certificate for an existing private key, you have to specify the password of the private key. keytool will attempt to verify the signer of the certificate by constructing a certificate chain to a trusted certificate. If a chain cannot be established, the import will fail – you will not be asked to verify the authenticity of the certificate. To have a successful import of an answer to a Certificate Signing Request, you have to trust the Certificate Authority which issued the certificate. If your Certificate Authority is one of the popular ones (for example, VeriSign or Thawte) you could rely on the certificates in the default truststore of the JVM (java.home/lib/security/cacerts) by using the "-trustcacerts" option of keytool:

```
keytool -importcert -alias myserverkey -keystore mykeystore -file  
myserverkey.der -storetype JCEKS -keystore mystore.jck  
-storepass mystorepass -trustcacerts
```

Extend the validity of a certificate using keytool

Suppose you have a JCEKS keystore called mystore.jck that includes an expired (or about to expire) self-signed certificate whose alias name is "myserverkey". The keystore has the associated private key in it. Assume that the password for the keystore is "mystorepass" and the password for the private key is "mykeystore". Now, if you want to extend the validity of this certificate by another 365 days, you can run the following command using keytool:

```
keytool -selfcert -v -alias myserverkey -keystore mykeystore -validity 365  
-storetype jceks -keystore mystore.jck  
-storepass mystorepass
```

The above operation will generate a new self-signed certificate, that has the same DN, SIGALG, KEYS as the original certificate but has a new SERIAL NUMBER and VALIDITY period.

Note: The generated new certificate will automatically replace the original one.

So if you need the original one later for reference or for any reason, you must keep a copy of the original keystore before doing the certificate extension explained above.

Note that this works only for self-signed certificates. It actually generates a new self-signed certificate for the public key, so you need to export it and update the truststores of the SSL parties that you are going to communicate with.

Work with keys stored in PFX/PKCS#12 files

As far as Java is concerned PKCS#12 is just another type of keystore (like JCEKS and JKS). To work with PKCS#12 keystores just set the "-storetype" option of keytool to "pkcs12". For example the following command lists the content of a mystore.p12 PKCS#12 file with password "mystorepass":

```
keytool -list -storetype pkcs12 -keystore mystore.p12 -storepass mystorepass
```

Create a keystore file

You don't need to create keystore files before you use them - keytool will automatically create a new keystore file, when it needs to write something to a file that does not exist. For example, if you generate a new key or import a certificate in a non-existing keystore, keytool will create the keystore file first.

Run keytool in FIPS mode

To run keytool in FIPS-compliant mode use the "-providerClass" option on each command like this:

```
keytool -list -storetype JCEKS -keystore mystore.jck -storepass mystorepass  
-providerClass com.ibm.crypto.fips.provider.IBMJCEFIPS
```

Secure Sockets Layer (SSL) Support

SSL is an important foundation for many Tivoli Directory Integrator security features. You need a working-level knowledge of SSL in order to fully exploit the capabilities in TDI.

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

- Connectors
 - AD Change Detection Connector
 - Axis Easy Web Service Server Connector
 - Axis2 Web Service Server Connector
 - Domino Change Detection Connector
 - Domino Users Connector
 - DSML v2 SOAP Server Connector
 - FTP Client Connector
 - HTTP Server Connector
 - IDS Changelog Connector
 - IBM Mq Series Connector
 - JMS Connector
 - JMS Password Store Connector
 - JNDI Connector
 - LDAP Connector
 - LDAP Group Connector
 - LDAP Server Connector
 - Lotus Notes connector
 - Mailbox Connector
 - Sun Directory Change Detection Connector
 - LDAP Connector
 - TADDM Change Detection Connector

- TADDM Connector
- TCP Connector
- TCP Server Connector
- TPAE IF Change Detection Connector
- Web Service Receiver Server Connector
- zOS LDAP Changelog Connector

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://download.oracle.com/javase/6/docs/technotes/guides/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 – “Server SSL configuration of TDI components” and “Client SSL configuration of TDI components” on page 96.

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 IBM Tivoli Directory Integrator must be defined. For information on keystores and truststores, see the guide at <http://download.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html> The following steps are required to enable SSL support for IBM Tivoli Directory Integrator as a server:

Note: RMI is enabled by default in the Tivoli Directory Integrator server. Properties for server authentication carry the default keystore property values.

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 *TDI_install_dir/jvm/jre/bin*, or *TDI_install_dir/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 *TDI_install_dir/etc/global.properties* file for the keystore file location, keystore file password and keystore file type.

```
## client authentication
javax.net.ssl.keyStore=serverapi\testadmin.jks
{protect}-javax.net.ssl.keyStorePassword=administrator
javax.net.ssl.keyStoreType=jks
```
5. Enable SSL for the clients (for example, using https in the Web browser).
6. 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 SSLSocket 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 `SSLocket` or the `SSLSocket` Java class and make it use a certificate different from the default.
4. See section "Certificates for the TDI Web service Suite" on page 135 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, see the guide at <http://download.oracle.com/javase/6/docs/technotes/guides/security/jsserrefguide.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.

Note: These four lines (comments starting with #) are no longer needed for client and server authentication to the Tivoli Directory Integrator server. Stores that belong to Tivoli Directory Integrator are set up to be used by default. This is part of enabling Remote Method Invocation (RMI) by default.

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

6. Enable SSL for the Connectors.
7. 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 135 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, Tivoli Directory Integrator needs a keystore as well. In this case the keystore can be defined just like it is defined when TDI is used a server – see the section "Server SSL configuration of TDI components" on page 95.

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 automatically sends 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 steps to configure SSL for IBM Tivoli Directory Integrator and Microsoft Active Directory:

1. Install Certificate Services on the Windows Server and an Enterprise Certificate Authority in the Active Directory Domain. Details are available at <http://windowsitpro.com/article/articleid/14923/how-do-i-install-an-enterprise-certificate-authority.html>. 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. Right-click, under the domain, **Domain Controllers**.
 - c. Select **Properties**.
 - d. Select the **Group Policy** tab, and 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.

Note: 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 computer 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 computer 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**.
- The next page automatically highlights the CA certificate.
- e. Click **Download CA certificate**
- A new download window opens.
- f. 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 found in *root_directory/jvm/jre/bin*, or *root_directory/jvm/bin*, depending on your platform.

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 like this script:

```
C:\Program Files\IBM\TDI\V7.1.1\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\V7.1.1\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. Enable SSL for your LDAP connector.
 - a. Go to the LDAP Connector configuration window.
 - b. Change **LDAP URL** to port 636.
 - c. Check **Use SSL**.
8. Restart IBM Tivoli Directory Integrator.

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

Summary of properties for enabling SSL and PKCS#11 support

You can configure SSL properties for server authentication, client authentication, and PKCS#11 support. See “Using cryptographic keys located on hardware devices” on page 163 on Public Key Cryptography Standards (PKCS).

Table 14. SSL Server Authentication

Property	Default value	Description
<code>javax.net.ssl.trustStore</code>	<code>serverapi\testadmin.jks</code>	Location of the truststore files.
<code>{protect}-javax.net.ssl.trustStorePassword</code>	administrator (encrypted by default)	truststore password.
<code>javax.net.ssl.trustStoreType</code>	<code>jks</code>	Type of the truststore.

Table 15. SSL Client Authentication

Property	Default value	Description
javax.net.ssl.keyStore	serverapi\testadmin.jks	keystore files location.
{protect}-javax.net.ssl.keyStorePassword	administrator (encrypted by default)	keystore password.
javax.net.ssl.keyStoreType	jks	Keystore type.

Table 16. PKCS#11 Support

Property	Default value	Description
com.ibm.di.pkcs11cfg	etc\pkcs11.cfg	Use this to specify the path of the configuration file required to initialize the IBM PKCS11 implementation provider. Added in TDI 7.0.
com.ibm.di.server.pkcs11	false	Use pkcs11 compliant crypto devices for ssl. Added in TDI 7.0.
com.ibm.di.server.pkcs11.library	none	Specify the path to the PKCS11client library. Added in TDI 7.0.
com.ibm.di.server.pkcs11.slot	none	Specify the slot number of the device.
{protect}-com.ibm.di.server.pkcs11.pass	none	Access the pkcs11 compliant crypto device using this password. Encrypted by default. Added in TDI 7.0.

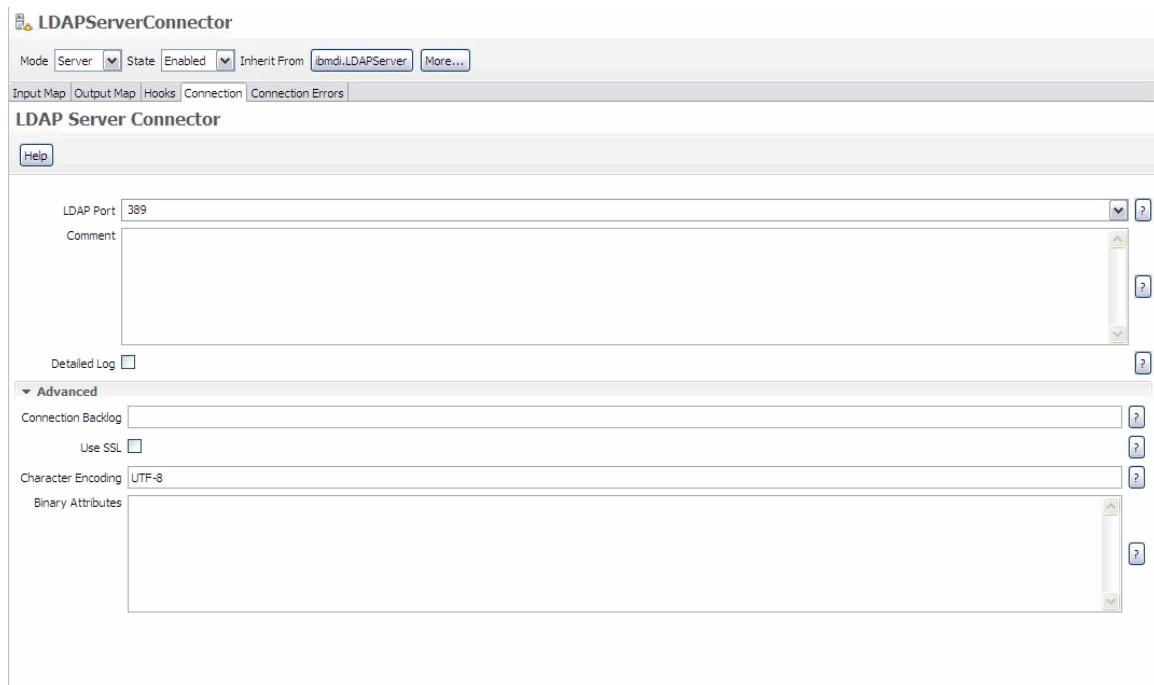
SSL example

In order to demonstrate how Tivoli Directory Integrator 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 the section called "Generate a public/private key pair and a self-signed certificate".
2. Set the keystore details in `global.properties` or `solution.properties` as described in the "Server SSL configuration of TDI components" on page 95 section.
3. Select **Use SSL** on the Connector GUI configuration window. You may need to expand the Advanced section to make the parameter visible.



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 96 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 Tivoli Directory Integrator Server; this is discussed in "Tivoli Directory Integrator Server Instance Security" on page 121. 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 Tivoli Directory Integrator Server by means of an access layer called RMI.

Note: The "remote Server" could very well be running on the same computer as the client application, for example if you start up a Server instance on your local computer and then access it using the Remote API through the loopback address, 127.0.0.1. 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
- Cycling through AssemblyLines

Note: Increasing needs for remote server access for each running Tivoli Directory Integrator server have resulted in a change from local access by default to remote access by default. As of 7.1.1, the remote server API is enabled by default. Prior to 7.1.1, the server API was enabled only for local access by default, where local access means access from the same Java Runtime Environment (JRE). To ensure security, remote access requires SSL client authentication. SSL access using client authentication is provided with the sample keystore and truststore deployed with TDI.

The Server API itself is documented in the Tivoli Directory Integrator Java API documentation (*TDI_install_dir/docs/api*; you can launch a browser to display this documentation by selecting **Help -> Welcome -> JavaDocs** 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 V7.1.1 Reference Guide*.

The Configuration Editor uses the Remote API to talk to the server you use to test-run your solutions. If this Tivoli Directory Integrator server is running on the same machine, it is often called the "local development server". For setups where the deployment platform does not support the Configuration Editor (for example, z/OS or i5/OS), you can run the development server on the deployment server, and the Configuration Editor on a supported platform like Windows (this way of running we call the "Remote Configuration Editor"). This design provides a uniform interface for both remote and local Config files. For some aspects of the Configuration Editor talking to a remote deployment server, see "Using the Remote Configuration Editor" on page 131.

The Server API is configured through a set of server properties (see "Configuring the Server API"). 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 105 (which secures the transport channel to a remote TDI Server),
2. "Server API authentication" on page 106 (which handles the client authentication to a TDI Server),
3. "Server API Authorization" on page 114 (which handles the client authorization to a TDI Server, that is, what the client is allowed to do once authenticated).

Configuring the Server API

The relevant properties are:

Property	Default value	Description
<code>api.on</code>	<code>true</code>	If set to <code>true</code> , 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 <code>api.on</code> is set to <code>true</code> .

Property	Default value	Description
<code>api.audit.on</code>	<code>false</code>	If set to <code>true</code> , the audit feature is turned on. If it is set to <code>true</code> , an audit entry is created at each audit point, even if the audit notifications are suppressed.
<code>api.user.registry</code>	<code>serverapi/registry.txt</code>	If configured, specifies the Server User Registry file name.
<code>api.user.registry.encryption.on</code>	<code>false</code>	If set to <code>true</code> , the Server API decrypts the Server User Registry file on startup.
<code>api.remote.on</code>	<code>true</code>	If set to <code>true</code> , 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.
<code>api.remote.ssl.on</code>	<code>true</code>	If set to <code>true</code> , SSL with client and server authentication is used on RMI connections of the Server API and its JMX layer; and the Server API uses the Server certificate and private key (the one specified through the <code>api.keystore</code> and <code>api.key.alias</code> properties) for SSL connections. RMI clients must trust that certificate. If set to <code>false</code> , 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 <code>api.remote.nonssl.hosts</code> property; if <code>api.remote.nonssl.hosts</code> is empty, only connections from the local host are accepted.
<code>api.remote.ssl.client.auth.on</code>	<code>true</code>	If set to <code>true</code> , the SSL client authentication for remote Server API is switched on.
<code>api.remote.naming.port</code>	1099	If specified, the port on which the RMI registry listens for requests.
<code>api.remote.server.ports</code>	8700-8900	The range of ports that may be used by the various RMI services. Entered as a comma separated list allowing hyphen-marked intervals (for example 1, 3-5, 10). Port numbers must be > 0 and <= 65536. The server will use these ports to listen for incoming RMI service requests, in addition to listening on the ports defined by other properties. For outgoing RMI service requests, random port numbers may be used.
<code>com.ibm.di.default.bind.address</code> *		The default bind address for the whole TDI Server - the components and the Server API. * means bind to all available network interfaces. Missing or invalid value binds to all interfaces, too. Only one IP address should be provided as value. This property will affect all Server Connectors that create a Server Socket.
<code>api.remote.bind.address</code>	*	The bind address for the RMI Server API. Overrides the <code>com.ibm.di.default.bind.address</code> property. * means connect to all available network interfaces. Missing or empty value means fall back to <code>com.ibm.di.default.bind.address</code> . Only one IP address should be provided as value.
<code>api.truststore</code>	<code>testserver.jks</code>	If specified, the keystore file that contains the public certificates of all remote users of the Server API.
<code>{protect}-api.truststore.pass</code>	server (encrypted by default)	If specified, the password for the keystore file named through the <code>api.remote.server.truststore</code> property.
<code>api.remote.nonssl.hosts</code>		If specified, shows a list of IP addresses to accept non-SSL connections from (host names are not accepted). Use space, comma or semicolon as a delimiter between IP addresses. This property is only taken into account when <code>api.remote.ssl.on</code> is set to <code>false</code> .

Property	Default value	Description
<code>api.jmx.on</code>	<code>false</code>	If set to <code>true</code> , 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.
<code>api.jmx.remote.on</code>	<code>false</code>	If set to <code>true</code> , 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.
<code>api.config.folder</code>	<code>configs</code>	If set to <code>TDI_root/configs</code> , only the configuration files placed in this folder can be edited through the Server API.
<code>api.config.lock.timeout</code>	0	If set to <code>0</code> , there is no timeout.
<code>api.custom.method.invoke.on</code>	<code>false</code>	The ability to use <code>Session.invokeCustom()</code> methods can be turned on or off (the default is false, or off). If the value of this property is set to <code>true</code> then users can use these methods, otherwise an Exception will be thrown.
<code>api.custom.method.invoke.allowed.classes</code>		If specified, gives the list of classes that can be invoked directly by the Server API methods for custom method invocation (<code>Session.invokeCustom(...)</code>). This property is only taken into account if <code>api.custom.method.invoke.on</code> is set to <code>true</code> . The classes in this list must be separated by a space, a comma or a semicolon.
<code>api.custom.authentication.</code>	[ldap]	If specified, points to a JavaScript text file that contains custom authentication code. To enable the built-in LDAP/JAAS Authentication mechanism, set this property to [ldap]/[jaas].
<code>com.ibm.di.server.id</code>		If specified, contains the server ID. Assign a unique value for each server from the set of servers that are running on the same IP and Port.
<code>api.config.load.timeout</code>	2	If specified, contains the serverapi config load timeout value in minutes. Added in Tivoli Directory Integrator v7.0.
<code>api.notification.suppress</code>	<code>di.server.api.authenticat</code>	If specified, gives a list of server notification types that you want to suppress. Notifications of suppressed types are not propagated by the notifications framework. Notification types in the list are separated by spaces. You can include wildcards. Example: <code>api.notification.suppress=di.al.*</code> <code>di.ci.start</code> The above example suppresses all AssemblyLine related notifications as well as notifications for starting a configuration instance. If the property is missing or is empty, no notifications are suppressed. Added in Tivoli Directory Integrator v7.0.
<code>api.client.ssl.custom.properties</code>	<code>true</code>	Enables custom SSL properties for Server API clients. If true, the <code>api.client.*</code> properties will be used by Server API clients. Otherwise the default <code>javax.net.ssl.*</code> properties will be used. Added in Tivoli Directory Integrator v7.1.
<code>api.client.keystore</code>	<code>serverapi/testadmin.jks</code>	Keystore for Server API clients.
<code>api.client.keystore.pass</code>	<code>administrator</code>	Password for the keystore specified by the "api.client.keystore" property.
<code>api.client.keystore.type</code>	<code>jks</code>	Type of the keystore file specified by <code>api.client.keystore</code> ; optional property, if not specified the default keystore format for the JVM will be used
<code>api.client.key.pass</code>	<code>administrator</code>	Password for the private key. The key is located in the keystore specified by the "api.client.keystore" property.

Property	Default value	Description
<i>api.client.truststore</i>	serverapi/testadmin.jks	Truststore for Server API clients.
<i>api.client.truststore.pass</i>	administrator	Password for the truststore specified by the "api.client.truststore" property.
<i>api.client.truststore.type</i>	jks	Type of the keystore file specified by <i>api.client.truststore</i> ; optional property, if not specified the default keystore format for the JVM will 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 Tivoli Directory Integrator Server. What should be noted though is that the way these Java system properties are set might be different. In Tivoli Directory Integrator these properties are normally set by editing the *global.properties* or *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.

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 computer. This VPN-assigned IP address needs to be specified in an RMI Java system property. If the Server API client is the Remote Configuration Editor, then this property can be set in *global.properties* or *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.

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 Configuration Editor through a network connection
- Access the Server API from Tivoli Directory Integrator components running in a remote Tivoli Directory Integrator server (remote Server API access). Examples of such components are:
 - System Queue Connector
 - Server Notifications Connector
 and so on.
- 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.
- Access the Server API from non-TDI Java applications. For this to work:
 - Java 6 or higher is required on the client side.
 - The following jar files must be included in the CLASSPATH of the remote side:
 - jars/common/diserverapi.jar
 - jars/common/diserverapirmi.jar

- jars/3rdparty/others/log4j-1.2.16.jar
- jars/common/miconfig.jar
- jars/common/miserver.jar
- jars/common/mmconfig.jar
- jars/common/tdiresource.jar
- jars/3rdparty/IBM/icu4j-4_4_2.jar
- jars/3rdparty/IBM/jlog.jar

You can copy these jar files from the Tivoli Directory Integrator installation.

- If custom non-TDI objects are used in the solution being implemented with the Server API (for example as Attribute values of an Entry that is transferred over the wire) the corresponding Java classes have to be available on the client side as well. These classes must be serializable and they have to be included in the CLASSPATH of the client JVM.

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. Tivoli Directory Integrator 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 must be configured. There are two options for configuring these. Which of them is 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 Tivoli Directory Integrator 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.keystore.type`** – specifies the type of the keystore file specified by `api.client.keystore`; optional property, if not specified the default keystore format for the JVM will be used
- **`api.client.key.pass`** – specifies the password of the private keystored in the keystore file contained in `api.client.keystore`; if this property is missing, the password specified by `api.client.keystore.pass` is used instead.
- **`api.client.truststore`** – specifies the keystore file containing the TDI Server public certificate.
- **`api.client.truststore.pass`** – specifies the password for the keystore file specified by `api.client.truststore`.
- **`api.client.truststore.type`** – specifies the type of the keystore file specified by `api.client.truststore`; optional property, if not specified the default keystore format for the JVM will be used

Use the Server API specific SSL properties when your client application is using the standard Java SSL properties. The standard Java SSL properties are properties used to configure 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.client.truststore=C:\TDI\serverapi\testadmin.jks
-Dapi.client.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 Tivoli Directory Integrator. 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.

You can specify these properties in `global.properties` or `solution.properties` when the client is a Tivoli Directory Integrator 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, and so on. When a local client establishes a local Server API session, the client has two options:

- Do not provide a username and 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, see "Server API Authorization" on page 114.
- Provide a username and password pair – in this case the Server API session is established only after the "username" supplied in the username and password pair is authorized according to the Server API Authorization logic described in the "Server API Authorization" on page 114 section. This option would normally be used when a specific user ID is needed for authentication – for demos, prototyping, and so on.

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 Configuration Editor, or a Java application wishing to access a TDI Server. For access of this kind there are the following methods of authentication to the TDI Server:

JAAS authentication

The Java Authentication and Authorization Service (JAAS) is supported as an authentication module for Tivoli Directory Integrator Server APIs. JAAS is a set of APIs that enables services to authenticate and enforce access controls upon users. The JAAS authentication is facilitated by the Tivoli Directory Integrator Server API. No changes are required on the TDI Server API clients such as CLI and AMC in order to use the JAAS authentication module.

In order to use JAAS authentication, you must configure the appropriate properties in `global.properties` or `solution.properties` and the JAAS Logon should be installed.

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, that is, 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 135.

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 116. 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 Tivoli Directory Integrator 7.1.1. 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 requires client authentication within the SSL handshake (the TDI 6.0 mechanism for SSL-based authentication). SSL client authentication for Tivoli Directory Integrator Server API does not depend on whether a username and password pair is supplied. This means that if no username and password pair is supplied, the TDI 6.0 mechanism for SSL-based authentication is used. And if a username and password pair is supplied then the client still needs to send its SSL certificate for authentication, but the User ID for authentication (and at a later step authorization) is 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 Tivoli Directory Integrator 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 and password based authentication. This hook allows bundlers/deployers to write customized JavaScript code, which given a username and 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 is executed for each username and password based authentication request.

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

- `userdata.username` – contains the name of the user requesting authentication
- `userdata.password` – contains the password provided by the user

The script is free to perform whatever checks and authentication actions it needs. It returns 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 is provided by the `AuthenticationException` thrown by the `ServerAPI` on unsuccessful authentication.

The authentication script has access to the main script object. It can be used for logging custom messages in the Tivoli Directory Integrator 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_install_dir/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, and so on) 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" tries to bind to an LDAP Server with the specified username and password. If the bind operation is successful, the script indicates a successful authentication, otherwise the authentication is 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 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 leverage your existing LDAP infrastructures that 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 and password authentication. For more information

on username and password authentication 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 can only use the TDI 6.0 SSL-based authentication mechanism. The Tivoli Directory Integrator 7.1.1 username and password authentication does not work. Set this property to "[ldap]" to enable the Tivoli Directory Integrator 7.1.1 built-in LDAP Authentication mechanism, like this:
api.custom.authentication=[ldap] All properties starting with "*api.custom.authentication.ldap.*" are 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 fails and the Server API is not started.

If this parameter is missing or is set to "false" the Server API logs the LDAP Authentication initialization error but the Server API is started. An attempt to initialize the LDAP Authentication module is 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 is used, otherwise SSL is not used.

api.custom.authentication.ldap.searchbase

Specifies the LDAP directory location where user searches is preformed. When this property is not specified user searches is not performed.

api.custom.authentication.ldap.admindn

Specifies an LDAP Server administrator distinguished name that is used for user searches. When this property is not specified anonymous bind is 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 are not performed. An example setting of this property would be:

api.custom.authentication.ldap.userattribute=cn

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

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

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

When **api.custom.authentication.ldap.admindn** is not provided a JNDI search context is 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 is performed with the distinguished name of the returned entry and the password provided for authentication. The authentication succeeds 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.

LDAP Group Support: To ease administration, Tivoli Directory Integrator allows permissions to be configured for groups the same way as they are configured for users. You can set permissions in the User Registry using exactly the same syntax as you would for a user. The fact is that the User Registry does not care whether a security entity is a group or a user. The distinction between users and groups is drawn during the authentication process.

Group membership is configured in the LDAP directory, against which Tivoli Directory Integrator authenticates users. If a user is a member of some LDAP group, all permissions for that group are automatically inherited by the user when the user is authenticated. Group support is disabled by default, so you must turn it on.

The system properties that are related to LDAP group support are:

`api.custom.authentication.ldap.groupsupport`

This is an optional property – a boolean flag. If this property is missing, the default value "false" is used. Specifies whether group membership is resolved when authenticating users. If the group membership is resolved, it is taken into account during authorization.

`api.custom.authentication.ldap.usermodelmembershipattribute`

This property is required only if `api.custom.authentication.ldap.groupsupport` is set to true. Specifies the name of the attribute of a user in LDAP that contains a list of the groups of which the user is a member.

`api.custom.authentication.ldap.usermodelmembershipattributecontent`

This property is required only if `api.custom.authentication.ldap.groupsupport` is set to true. Specifies how groups are named in the membership attribute of a user. For example, if the user's

membership attribute contains values that correspond to the "objectSID" attributes of groups, set this property to "objectSID". If the user's membership attribute contains distinguished names of groups, then set this property to "dn".

api.custom.authentication.ldap.groupnameattribute

This property is required only if **api.custom.authentication.ldap.groupsupport** is set to true. Specifies the name of a group's attribute in LDAP which corresponds to the way the group is named in the TDI User Registry. For example, if LDAP groups are addressed in the Tivoli Directory Integrator registry by their common name, then set this property to "cn". If the User Registry contains the distinguished names of the groups, then set this property to "dn".

api.custom.authentication.ldap.groupsearchbase

This property is required only if **api.custom.authentication.ldap.groupsupport** is set to true. Represents the LDAP directory context, where groups are searched.

api.custom.authentication.ldap.binaryattributes

This is an optional property – it represents a list of space-separated attribute names. Specifies attributes which have non-string syntax.

Active Directory example:

This example shows how to configure group support to work with an **Active Directory** server:

```
api.custom.authentication.ldap.groupsupport=true  
api.custom.authentication.ldap.usermodelshipattribute=tokenGroups  
api.custom.authentication.ldap.usermodelshipattributecontent=objectSID  
api.custom.authentication.ldap.groupnameattribute=sAMAccountName  
api.custom.authentication.ldap.groupsearchbase=DC=mytestadserver,DC=com  
api.custom.authentication.ldap.binaryattributes=objectSID tokenGroups
```

The 'tokenGroups' attribute is a calculated attribute that exists for all users in Active Directory.

It contains a collection of the Security Identifiers (SIDs) for all security groups that the user is a member of.

This collection contains only security groups (distribution groups, used for e-mail, are not included) and it contains all security groups including nested and primary groups.

The Security Identifiers are binary attributes so they must be set in the **api.custom.authentication.ldap.binaryattributes** property.

In the above example, groups are named by their "sAMAccountName" LDAP attribute in the TDI User Registry.

Tivoli Directory Server example:

This example shows how to configure group support to work with **Tivoli Directory Server**:

```
api.custom.authentication.ldap.groupsupport=true  
api.custom.authentication.ldap.usermodelshipattribute=ibm-allGroups  
api.custom.authentication.ldap.usermodelshipattributecontent=dn  
api.custom.authentication.ldap.groupnameattribute=dn  
api.custom.authentication.ldap.groupsearchbase=ou=mytestou,c=mytestcountry
```

For a given user entry, the "ibm-allGroups" operational attribute enumerates all static, dynamic and nested groups, to which that user has membership.

Notes:

1. Tivoli Directory Integrator determines group membership by directly examining the LDAP user entry (as opposed to indirectly determining membership by scanning through all groups). For this approach

to work correctly, the user entry must have an attribute that enumerates the groups, of which the user is a member. The group support works only with LDAP Servers that do support such a membership attribute on each user entry.

2. If you modify the group membership of a user, this does not affect existing Server API sessions. It is, however, reflected in sessions established after the modification.
3. Group support is currently provided only for LDAP authentication. There is no group support for JAAS authentication or authentication with custom JavaScript.
4. When SSL client authentication is enabled in the Server API, clients that do not specify a username are to be authenticated and authorized based on the owner of the SSL client certificate. If LDAP authentication with group support is also enabled (along with the SSL client authentication), group membership is resolved for the owner of the SSL client certificate.

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 and 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 also need `api.on=true`, `api.remote.on=true`, `api.remote.ssl.on=true`). The user is authorized by 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 do not work and the remote client application cannot 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” on page 116.

LDAP authentication

This was described in section “LDAP Authentication support” on page 108, 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 and 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 and password authentication

The remote JMX layer of the Server API does not support username and 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, that is, 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 and password authentication:

Server API authentication setup examples

Authentication configuration examples:

1. Non-SSL configuration and 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 and password supplied succeed only if they are invoked from the localhost or from 192.168.113.51 or 192.168.113.52.
- Authentication requests with username and password supplied succeed only if the ldap_auth.js successfully authenticates the user specified with the username and password parameters.
- Remote JMX clients are 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) and 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 and password supplied fail because neither SSL client authentication, nor host-based authentication is switched on.
- Authentication requests with username and password supplied succeed only 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.
- Remote JMX layer is not accessible. This is because SSL is turned on but SSL client authentication is not used.

3. SSL with client authentication and 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 and password supplied succeed when the SSL certificate of the client is present in the Server's truststore (or verifiable using the certificates in the truststore).
- Authentication requests with username and password supplied succeed only when the SSL client authentication is successful (the SSL certificate of the client is present in the Server's truststore) and the ldap_auth.js script successfully authenticates the user specified with the username and

password parameters. In this case, authorization is performed based on the username parameter from the username and 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 are authenticated when the SSL certificate of the client is present in the Server's truststore (or verifiable using the certificates in the truststore).

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 and password supplied succeed when the SSL certificate of the client is present in the Server's truststore (or verifiable using the certificates in the truststore).
- Authentication requests with username and password supplied do 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 are authenticated successfully only when the SSL certificate of the client is present in the Server's truststore.

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 116 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 or password based authentication – the user id is the username supplied in the username and 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:

<i>Read role: read</i> [list_of(configuration)]	<p>The <i>read</i> role allows the user read data from the Server's configuration(s). If no list of configurations is specified or the list is empty, the user is not allowed to read any configuration.</p> <p>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.</p> <p>When the list of configurations is not null/empty and does not specify * the user is allowed to read only the configurations specified.</p> <p>The <i>read</i> role does not grant permission to start processes (AssemblyLines) or apply any changes to the Server and its configurations. For example:</p> <pre>[ROLE]:read [CONFIG]:*</pre>
<i>Execute role: execute</i> [list_of(configuration)[list_of(AssemblyLines)]]	<p>The <i>execute</i> role gives the user permissions to execute AssemblyLines.</p> <p>If no list of configurations is specified or the list is empty, the user is not allowed to execute any AssemblyLine from any configuration.</p> <p>A special value * (asterisk) can be specified for the list of configurations and this means that the user is allowed to execute all AssemblyLines from all configurations.</p> <p>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:</p> <ul style="list-style-type: none"> • If a list of AssemblyLines is not specified, the user is not allowed to execute any AssemblyLine from this configuration. • If a special value * (asterisk) is specified for the list of AssemblyLines, 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. <p>For example:</p> <pre>[ROLE]:execute [CONFIG]:C:/TDI/rs.xml [AL]:* [CONFIG]:C:/TDI/prototype.xml [AL]:TestAssemblyLine</pre>
<i>Admin role: admin</i>	<p>The <i>admin</i> role allows the user to execute all Server API calls in every possible context.</p> <p>A user with <i>admin</i> role is allowed to read and modify configurations, to load new configurations, to execute AssemblyLines, to read and modify server parameters.</p> <p>For example:</p> <pre>[ROLE]:admin</pre> <p>Note:</p> <p>Admin role is required to use the Remote Configuration Editor. Also see "Using the Remote Configuration Editor" on page 131.</p>

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.

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 *api.key.alias* property from the keystore specified by the *api.keystore* property. The encryption algorithm employed is Asymmetric RSA encryption or decryption; that is why the "Example Server certificate creation" on page 135 specifying the RSA algorithm, which is the default algorithm of the "The TDI Encryption utility" on page 126 provided with Tivoli Directory Integrator 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 used.
2. The authorization against the user registry is not optional. Currently the Tivoli Directory Integrator 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>
    ...
    [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]	This tag takes no arguments, and serves as an opening bracket for the tags below; a [USER] and [ENDUSER] pair of tags, each placed on a single line, provide the definition of a single user in the registry file. There can be multiple pairs of this type, 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 117 from the truststore 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>	This tag specifies a role for the user. Possible roles are: read , execute or admin . 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.

Tag	Argument
[CONFIG]:<config_id>	This tag specifies the identifier of a Tivoli Directory Integrator configuration, the absolute file path of the configuration. Relative file paths are not recognized. This tag is subordinate to a [ROLE]: tag, and the tag specifies a configuration for the role given by the [ROLE]: tag. 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, the list of configurations for the corresponding role definition is empty.
[AL]:<assembly_line_name>	This tag specifies an AssemblyLine name. 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, the list of AssemblyLines for the corresponding configuration ID is empty.

The following text 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]:*
[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]:*
[ENDUSER]
```

This set of Identity Registry entries implies the following constraints:

- "Stan" is an administrator according to this registry file, 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", John can execute all AssemblyLines.
 - From "prototype.xml" John is only allowed to execute the AssemblyLine named "TestAssemblyLine".
- Peter can only execute all AssemblyLines from the "rs.xml" configuration.

Note: The **keytool** and/or the **Ikeyman** utility can be used to obtain the user ID from the truststore file.
The following command line prints all users from the truststore 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 prints something like the text below for each user certificate:

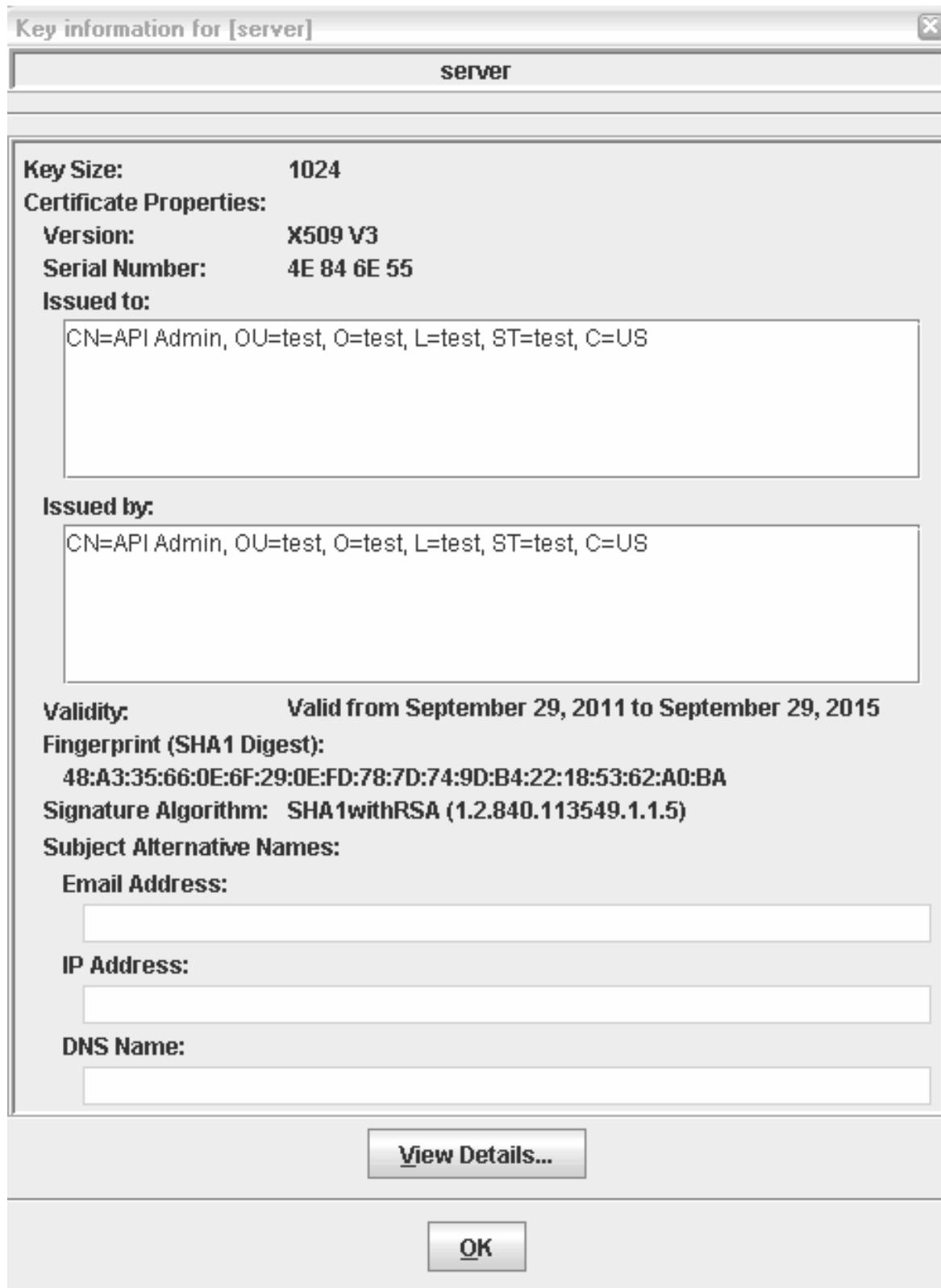
```
Owner: CN=Stan, OU=TDI, O=IBM, C=US
Issuer: CN=Stan, OU=TDI, O=IBM, C=US
Serial number: 408f6a34
Valid from: 4/28/04 11:24 AM until: 7/27/04 11:24 AM
Certificate fingerprints:
MD5: F6:EF:81:8B:4C:0F:10:E4:A0:16:99:AB:42:29:70:8B
SHA1: FE:37:62:8B:42:2F:54:F8:F6:F3:FC:A1:DD:7D:2A:51:9A:85:09:02
```

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 looks like:

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

An alternative way to obtain the user ID from the truststore file is to use Ikeyman in the following way:

1. Start Ikeyman (or select **Key Manager** from the toolbar).
2. From the **Key Database File** menu click **Open....**
3. In the **Open** field, set the appropriate values and click **OK**.
4. In the **Password** field, enter the password for the truststore file.
5. Click on the certificate you are interested in.
6. Click the **View/Edit...** button. A window opens which contains information on the subject's DN (user ID).



Server Audit Capabilities

The Tivoli Directory Integrator Audit Component enables the TDI Server to audit events such as authentication and authorization in the Server API.

Notifications are generated when authentication and authorization (auth*) events occur. Audit data is packaged into an Entry and provided as user data in the notification. The "Audit Service" consists of a separate Audit config that is automatically loaded by the TDI server. The Audit config contains auto-started Audit AssemblyLines. The Audit ALs iterate on the notification connector using suitable filters. Tivoli Directory Integrator users can even generate "user defined notifications" if they want to create audit events from within their own code.

Tivoli Directory Integrator auditing contains two main parts:

- A way for generating the necessary audit information
- An "Audit service" for handling existing audit data

Generating necessary audit information is implemented by creating Tivoli Directory Integrator Entries on each audit point in the Server API, and by broadcasting these Entries wrapped in a notification. For this purpose a new class is presented in the Server API (com.ibm.di.api.APIAuditor), that generates the Entry, attaches the Entry as UserData to a notification, and sends it to all interested listeners.

The "Audit Service" is the main consumer of the audit notifications. The Audit Service is a config consisting of several ALs that iterate on the Notification Connector. Using different filters can register to a variety of notification types.

Auditing scope

The TDI audit capability follows only what people do, and does not follow Server events in general. There is a difference between a user being authorized to perform a task (stop an AL) and the task actually being performed (AL is terminated). Being authorized is an authorization event and the performing of a legal action, like stopping an AL, is a Server event. When a user instructs an AL to stop and the AL terminates, an authorization event is paired with a Server event. At other times, a Server event occurs by itself, as when an AL completes naturally. Only events which involve direct user interaction are audited. This limits the default audit points to authentication and authorization events inside the Server API. Almost every method exposed by the Server API is protected by its own piece of authorization code. The Audit component does not try to send notifications for all authorization events, but selects a reasonable subset of authorization-guarded Server API methods. The principles for the selection are to audit all events that:

- Delete logs or tombstones
- Start or stop TDI entities such as configs, ALs, and the Server
- Replace the config instance configuration: replace the config instance configuration or the check-in configuration
- Allow the user to change vital TDI data: set external property, post a message in the System Queue, call custom Java code inside the Tivoli Directory Integrator JVM

Suppression of notifications

The Tivoli Directory Integrator Server API allows certain notification types to be suppressed for improved performance. The notification framework does not propagate suppressed events. If you try to broadcast an event of a type *suppressed*, the Server API does not issue an error. However, the suppressed event cannot reach any of the registered notification listeners. The list of suppressed event types is configured by a system property named:

`api.notification.suppress`

By default, all authentication and authorization events are suppressed:

`api.notification.suppress=di.server.api.authenticate di.server.api.authorize*`

The event types in the list are separated by spaces. Wildcards matching multiple event types are allowed. If the event type property is missing or is empty, no events are suppressed. You can suppress all custom notifications by typing:

`api.notification.suppress=user`

Note: Suppression affects the whole TDI Server and can result in suppression of all kinds of notifications. Even built-in notifications, such as an AssemblyLine starting or the Server shutting down, can be suppressed. Improper use of the suppression capabilities can interfere with the work of components that listen for notifications such as the Tombstone Manager and the Server Notifications Connector.

Sending notifications

Sending notifications uses a method in com.ibm.di.api.APIEngine:

```
public static void sendNotification (String type, String id, Object data, String configInstanceId)
```

This method creates a DEvent. By this means, a notification is delivered to every Listener registered to receive the a particular type of notification. Notification delivery parameters include:

Table 17. Notification Delivery Parameters

Parameter name	Definition
type	Notification event type.
id	Notification event ID.
data	Notification event UserData object in the form of a Java object with additional information.
configInstanceId	Notification ConfigInstance ID to which the notification is bound.

The com.ibm.di.api.APIEngine method throws DIException if the type parameter is null. Calls to the method can be invoked either:

- Locally, from the Tivoli Directory Integrator Server JVM. This type of access includes scripting in AssemblyLine hooks and also uses the API from new Connectors implemented in Java and deployed on the Tivoli Directory IntegratorServer
- Remotely, from another JVM (on the local or a remote network computer), through Remote Method Invocation (RMI). This type of access uses solutions that:
 - Connect remotely to Tivoli Directory Integrator
 - Manage processes within Tivoli Directory Integrator
 - Build business logic on top of Tivoli Directory Integrator
 - Are applications dedicated only to Tivoli Directory Integrator
 - Are applications that use Tivoli Directory Integrator to accomplish some of their goals

Tivoli Directory Integrator 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 100; instead, it focuses on the encryption algorithms used, and 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, as well as being used for SSL communication.

The system properties *api.keystore* and *api.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 stash file of the server, see section “Stash File” on page 122.

The RSA algorithm is used for encryption of files and property values. It is used as a block cipher where the block size is determined by the modulus component of the RSA key. Encryption is done in ECB (Electronic Codebook) mode. PKCS#1 Padding is applied separately on each block. Note that the same RSA key-pair, which is used for encryption of files, is also used for SSL communication with the Server. Tivoli Directory Integrator uses the RSA implementation from the IBMJCE security provider. All key sizes supported by that provider are also supported by Tivoli Directory Integrator. From Tivoli Directory Integrator v7.0, secret key ciphers can also be employed for encryption. RSA is used as the default for compatibility with earlier versions, but secret key ciphers are much faster and much more secure than public key ciphers.

DES and AES algorithms are used for encryption of password-protected configuration files. An encryption key (DES or AES) is derived from the UTF-8 binary representation of the password. The derived encryption key is 64 bit for DES and 128 bit for AES. ECB mode is used with no padding.

DES/AES keys are derived from passwords, when using password-protected configuration files. Apart from the above, the Tivoli Directory Integrator does not generate keys. Existing keys are loaded from an external key store. Key establishment and key store access are performed through the IBMJCE and IBMJSSE2 security providers. All key sizes and algorithms supported by those providers can be used with the Tivoli Directory Integrator.

Stash File

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

```
createtestash <keyStorePassword> [<keyPassword>] [<securityProviderClass>]
```

where *keyStorePassword* is the password of the keystore file specified by the *api.keystore* system property and *<keyPassword>* is the password of the Server's private key specified by the *api.key.alias* system property.

keyPassword is an optional parameter if no *<securityProviderClass>* parameter is specified. If *<keyPassword>* is not specified it is assumed that the Server's private key password is the same as the keystore's password. To use the utility with the *<securityProviderClass>* parameter, you must specify both previous parameters: *keyStorePassword* and *keyPassword*. If a security provider is specified then this provider is used for the cryptography.

The utility creates a stash file named "idisrv.sth" with the specified password(s) in the current directory.

Attention: IBM Tivoli Directory Integrator 7.1.1 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 Tivoli Directory Integrator Server that needs it.

Server Security Modes

The Tivoli Directory Integrator Server can run in two modes: **standard** and **secure**.

Standard mode

When run in standard mode, the Server does 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 encrypts all configurations it saves on the disk using PKI encryption. In secure mode the Server can only read and load encrypted configurations. When the system property *com.ibm.di.server.securemode* is set to "true", the Server runs 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 runs 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 compatibility with earlier versions. 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 PKI encryption. If you specify a password when the Server is run in secure mode, an error message is displayed.

Working with encrypted TDI configuration files

Introduction

To provide confidentiality of data, Tivoli Directory Integrator (TDI) can encrypt configuration files, property values in properties files, server user registry files and JavaScript files.

TDI encryption involves a cryptographic transformation that uses a key or a key-pair. The key/key-pair needs to be hosted in a keystore file.

The cryptographic transformation can be either public-key encryption or secret key encryption. By default TDI uses public key encryption. (The secret key encryption option has been introduced in TDI 7.0. Before that only public key encryption was supported.)

See:

Public key encryption uses a key-pair that consists of a public key and a private key. The public key is used for encryption and the private key is used for decryption. Currently only the RSA cipher is supported for public key encryption. Public/private key pairs can be generated and managed using the standard JRE utilities `keytool` and `Ikeyman`. See “Manage keys, certificates and keystores” on page 89 for more information on managing certificates with associated public and private keys.

TDI data encryption is configured by the following system properties (these can be set in `global.properties` or `solution.properties`):

- `com.ibm.di.server.encryption.keystore` : the keystore file that contains the key/key-pair for encryption
- `com.ibm.di.server.encryption.keystoretype` : the type of the keystore file
- `com.ibm.di.server.encryption.key.alias` : the alias of the key/key-pair in the keystore
- `com.ibm.di.server.encryption.transformation` : the name of the encryption transformation; see remarks below

The password of the keystore and the password of the key/key-pair itself (if different from the keystore password) are specified in the Server’s “Stash File” on page 122. (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.)

The name of the transformation can be either RSA or some secret key transformation (for example, AES/CBC/PKCS5Padding). More detailed discussion of what is in a transformation name can be found at http://www.ibm.com/developerworks/java/jdk/security/60/secguides/JceDocs/api_users_guide.html#trans; general information about Java Security (which is what Tivoli Directory Integrator uses) can be found at <http://www-128.ibm.com/developerworks/java/jdk/security/60/secguides/jsse2Docs/JSSE2RefGuide.html>.

Notes:

1. The "com.ibm.di.server.encryption.*" properties affect not only encryption of configurations, but also encryption of property files, JavaScript files and the Server API User Registry.
2. If you change the encryption key and/or the encryption transformation, the Server cannot decipher previously encrypted files. To work around this problem, decrypt the old files with the old key (you must have the old key available in order to do so) and encrypt them with the new key. Encryption and decryption of files can be done with the cryptoutils tool.
3. The standard RSA algorithm has a restriction on the length of data it can work on. TDI uses a custom scheme that splits input data into small enough equally sized blocks and encrypts each of them separately.
4. Data encrypted with RSA result in different cipher-texts on different encryption runs. This effect is a feature of the PKCS#1 padding scheme used with RSA.
5. A secret key (symmetric) cipher can be either a block cipher or a stream cipher. Stream ciphers encrypt the bits of the message one at a time, and block ciphers take a number of bits and encrypt them as a single unit (a block). Block ciphers (for example, AES) use a feedback mode (so that patterns in the plain-text are not preserved in cipher-text) and a padding scheme (to allow encryption of data, whose length is not multiple of the block size of the cipher). Stream ciphers (for example, RC4) do not use a feedback mode and a padding scheme.
6. If the transformation involves a block cipher, it must use some padding scheme (for example, "PKCS5Padding"), otherwise the Server not be able to encrypt data whose length is not multiple of the block size of the cipher. (Stream ciphers do not use padding, so they are not affected by this restriction.)
7. The algorithm of the key/key-pair must match the algorithm in the specified transformation. For example if the transformation is RSA then an RSA key-pair must be provided; if the transformation is DES/ECB/PKCS5Padding, you must provide a DES key. You can generate a new secret key using the keytool utility, see "Manage keys, certificates and keystores" on page 89.
8. JKS keystores do not support secret keys, so you should use some other keystore type such as JCEKS if you want to use a secret key encryption.
9. When using a block cipher in a feedback mode that requires an initialization vector (IV), encrypted data is prefixed with the initialization vector as plaintext. The IV does not have to be kept secret but must be unpredictable. That is why a random IV is generated for each piece of data that is being encrypted. Generation of random data can sometimes be resource-intensive, so you may wish to consider a non-IV feedback mode (ECB) if performance is an issue.
10. Which secret key transformations are supported for encryption depends on the capabilities of the Java security provider. By default TDI uses the IBMJCE provider. Supported block ciphers are: DES, AES, DESede (Triple DES), Blowfish and RC2. They can be used in any of the following feedback modes – ECB, CBC, CFB, OFB, PCBC. The only available padding scheme is "PKCS5Padding". The MARS block cipher should not be used for encryption, because it does not support padding (<http://www-128.ibm.com/developerworks/java/jdk/security/50/secguides/JceDocs/api/com/ibm/crypto/provider/Mars.html>). Supported stream ciphers are RC4 and ARCFOUR (basically the same cipher under two different names). The SEAL stream cipher requires large keys (160 bit) so it can be used only after configuring unrestricted IBM SDK policy on the TDI JRE (<http://www.ibm.com/developerworks/java/jdk/security/60/#sdkpol>).

Separation of certificates for PKI Encryption and SSL

Creating an encrypted TDI configuration file from scratch

This is how you create an encrypted IBM Tivoli Directory Integrator configuration file from scratch.

Using the cryptoutils command line tool:

1. Create a normal un-encrypted TDI configuration file using the Configuration Editor.
2. Use the cryptoutils command line tool to encrypt this configuration file as described in the "The TDI Encryption utility" on page 126 section.

3. In order to run this encrypted configuration file you must 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

You can first decrypt the encrypted configuration file using the `cryptoutils` command line tool as described in the ""The TDI Encryption utility" on page 126" section. Then you can edit the decrypted configuration using the Configuration Editor and finally you can encrypt back the modified configuration file using the `cryptoutils` tool.

Standard TDI encryption of global.properties or solution.properties

The `global.properties` and `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 Tivoli Directory Integrator 7.1.1 is capable of encrypting this data.

All properties whose names are prefixed with `{protect}-` are PKI encrypted by the Server using the Server's public key. The Server's key is specified by the `com.ibm.di.server.encryption.key.alias` property from the keystore specified by the `api.keystore` property. For example, if you want to encrypt a property `com.ibm.di.server.encryption.keystore` 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 detects that this property has to be encrypted and it immediately overwrites 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 `global.properties` file might not be accessible for writing. In this case the server outputs a warning message that the file has not been written/encrypted.

Protecting the properties in `global.properties` or `solution.properties` is also accessible from the "Global-Properties" and "Solution-Properties" Property Stores accessible from the **Browse Server Stores** option in the Configuration Editor.

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.

Instead of using the server's default certificate, it is possible to encrypt properties in external property files using a specifically named certificate from the server's keystore.

For more information on encrypting properties stored in these files, see the "Standard TDI encryption of `global.properties` or `solution.properties`" 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}J8AKimpEtu3Bb10Vg55F/5d5v02kXWcNUWnCq3vINUc6K0719z9dEk3H430t2iT1dZTI6FSSVin9KsCyBLmgv+n84w7He1K13ro2dfmZbTYKMXuxGoqN9nL2V0vZoptNqzoWvs6IN/p3VkiIBtlao/9mEPEKuIwRnKtkQ89Bg=
```

The TDI Encryption utility

In the *TDI_install_dir/serverapi* directory you find a utility (*cryptoutils*) which enable you to decrypt and re-encrypt files, (for example, the Identity Registry file) such that you can edit the file manually.

The tool recognizes the following command-line parameters:

inputFile

{required} Specifies the file to be encrypted or decrypted.

outputFile

{required} Specifies the new file that is created with the resulting data after the encryption or decryption is done. If the file exists, it is overwritten.

mode {required} Specifies the mode in which the tool operate; it can be one of the following modes:

- *encrypt*: encrypt user registry
- *decrypt*: decrypt user registry
- *encrypt_config*: encrypt a TDI configuration file or a JavaScript file
- *decrypt_config*: decrypt a TDI configuration file or a JavaScript file
- *encrypt_props*: encrypt the values of all protected properties in a TDI properties file
- *decrypt_props*: decrypt the values of all protected properties in a TDI properties file

Note: User Registry files are encrypted differently from configuration and JavaScript files.

keyStore

{required} Specifies the keystore file which contains the key for encryption/decryption.

storepass

{required} Specifies the password of the keystore file.

alias {required} Specifies the alias of the encryption/decryption key in the keystore

keypass

{optional} Specifies the password of the encryption/decryption key; by default, the keystore password is used to access the key

transformation

{optional} Specifies the name of the cryptography transformation used for encryption/decryption; can be RSA or any secret key transformation (for example, AES/CBC/PKCS5Padding); the default is RSA.

storetype

{optional} Specifies the type of the keystore file (for example, JKS); this parameter is case-insensitive (JCEKS and jceks are equivalent); if this parameter is missing, the default keystore type of the JRE (configured by the "keystore.type" security property in the java.security file of the JRE) is used.

cryptoproviderclass

{optional} Specifies the Java security provider which is used for encryption/decryption (but not for keystore access); by default the providers from the security provider list of the JRE (configured in java.security JRE file) is used.

Examples:

Encrypt the User Registry

A TDI Server running in secure mode requires that the User Registry is encrypted with the Server key.

You can encrypt a plaintext User Registry file like this:

```
cryptoutils -input registry.txt -output registry.enc -mode encrypt -keystore ../testserver.jks -storepass server -alias server
```

Decrypt a TDI configuration

```
cryptoutils -input myconfig.enc.xml -output myconfig.xml -mode decrypt_config -keystore ../testserver.jks  
-storepass server -alias server
```

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 Configuration Editor. After modifying the configuration, it can be encrypted again, so that a TDI Server in secure mode can read and use it.

Encrypt a TDI configuration using a symmetric cipher (rather than the default "RSA")

```
cryptoutils -input myconfig.xml -output myconfig.enc.xml -mode encrypt_config -keystore ../server.jck  
-storepass server -alias server -transformation AES/CBC/PKCS5Padding -storetype jceks
```

The above command assumes that the keystore "server.jck" exists. That keystore is supposed to contain an AES secret key under alias "server".

Decrypt the global.properties file

The Tivoli Directory Integrator Server automatically encrypts the values of protected properties when reading the `global.properties` or `solution.properties` file.

You can decrypt all encrypted values in the `global.properties` file like this:

```
cryptoutils -input ../etc/global.properties -output ../etc/global.properties -mode decrypt_props  
-keystore ../testserver.jks -storepass server -alias server
```

Note: When the `cryptoutils` tool is used to encrypt and decrypt the User Registry, configuration files (see the “Server Security Modes” on page 122” section for details how the server treats encrypted configurations) or “Encryption of TDI Server Hooks” on page 130, it encrypts and decrypts a file as a whole.

On the other hand, the encryption/decryption mode for property files encrypts/decrypt only the values of the protected properties and not the whole file. Thus after encrypting a .properties file using `encrypt_props` mode, the property keys and the comments in the file are still readable by humans. For more information on protected properties see sections “Standard TDI encryption of `global.properties` or `solution.properties`” on page 125 and “Encryption of properties in external property files” on page 125.

TDI System Store Security

The Tivoli Directory Integrator System Store is the database or persistent layer where all the information which is required by a Tivoli Directory Integrator Server is persisted. Traditionally, this layer did not have any security around itself. Any user was able to access the System Store. However from Tivoli Directory Integrator 7.0, there is configurable security provided around the System Store.

In Tivoli Directory Integrator 7.0, the System Store by default is used in Network Mode. This way, a number of Tivoli Directory Integrator instances and other applications is able to access the System Store concurrently. In view of the System Store being available over the Network there is a need to have some security built around it in order to protect the data which is maintained by the Tivoli Directory Integrator Server.

Derby (previously known as Cloudscape) provides several ways to define the repository of users and passwords. To specify which of these services to use with your Derby system, set the property `derby.authentication.provider` to the appropriate value as discussed in the appropriate section listed below.

External Directory Service

A directory service stores names and attributes of those names. Derby uses the Java naming and directory interface (JNDI) to interact with external directory services that can provide authentication of users' names and passwords.

You can allow Derby to authenticate users against an existing LDAP directory service within your enterprise. LDAP (lightweight directory access protocol) provides an open directory access protocol running over TCP/IP. An LDAP directory service can quickly authenticate a user's name and password.

On configuring a set of properties defined by Derby you can start using the External Directory Service as a repository for user names and passwords.

User-defined class

The user defined class approach enables you to hook Derby to any other external authentication service other than LDAP.

Set `derby.authentication.provider` to the full name of a class that implements the public interface `org.apache.derby.authentication.UserAuthenticator`. By writing your own class that fulfills some minimal requirements, you can hook Derby up to an external authentication service.

The class that provides the external authentication service must implement the public interface `org.apache.derby.authentication.UserAuthenticator` and throw exceptions of type `java.sql.SQLException` where appropriate.

Built-in Derby Users

Derby provides a simple repository for storing the user names and passwords. For using this built-in repository the property `derby.authentication.provider=BUILTIN` should be set..

The Tivoli Directory Integrator System Store is using the Built-in repository for storing the user name and password. Since Tivoli Directory Integrator have only one user for accessing the System Store this is the most viable provider that can be used.

User Authentication:

The user authentication details deal with the authentication of users. The user authentication mechanism only authenticates if the user name is present in the mentioned repository (it can any one of the repositories which are mentioned above) and if the password is correct for the specified user. However if you want to have more control over the access rights, you can use the User Authorization mechanism provided by Derby.

The master switch for requiring that users be authenticated against provided parameters is the property `derby.connection.requireAuthentication` - the default is `TRUE`.

The access modes can be set using the property `derby.database.defaultConnectionMode=fullaccess`. This property sets the default access mode for all the users in the Derby repository. This property also defines the access level for the System Store user. The different access levels supported by Derby are `fullAccess`, `readOnly`, and `noAccess`. However if you want to have specific access modes for specific users, you can assign access using the properties mentioned below:

- `derby.database.fullAccessUsers=<usernames>` for allowing full access to users.
- `derby.database.readOnlyAccessUsers=<usernames>` for allowing read only access to users.
- `derby.database.noAccessUsers=<usernames>` for not allowing users to access the database.

The `usernames` should be a comma separated list of users for example

`derby.database.fullAccessUsers=sa, mary`

In the current version of Tivoli Directory Integrator we have only one user accessing the System Store. This user is required to perform all the operations on the System Store hence we have set the access mode to `fullAccess`.

Miscellaneous Config File features

The "password" configuration parameter type

The configuration parameters of a Tivoli Directory Integrator component in a Config can be "string", "number", "boolean", and so on. One of the available types is "password". If a configuration parameter is of type password, then the Configuration Editor shows its value in the component configuration window 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

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

Tivoli Directory Integrator provides a way to better protect passwords that are needed for its various components; it hides the passwords in a clear text configuration and provides default security for passwords that are stored. In order to do this component passwords are defined (stored and retrieved) in a default property store, instead of in the configuration file. In Tivoli Directory Integrator 7.1.1, 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 older versions of the product). 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 windows offered to the user. The configuration windows, or forms, contain descriptions of each parameter and its syntax. One type of syntax is *password* which causes the Configuration 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 is 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 is 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 project from the Configuration Editor
2. Right-click on the "Properties" folder in the navigation view and select "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 is used as the Password Property Store (right-click on the new properties store in the navigation view and select **Password Property Store**).
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 Configuration Editor.

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

```
<parameter name="ftpPass">@SUBSTITUTE{property.MyProps:ftpPass-38ae53e8779cf65}</parameter>
.....
<PasswordStore>MyProps</PasswordStore>
```

...and in the "MyProperties.properties" file there is a line like the following text:

```
{protect}-ftpPass-38ae53e8779cf65={encr}GVJC01A7VUiW=
```

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-38ae53e8779cf65".

Protecting attributes from being printed in clear text during tracing

Tivoli Directory Integrator 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 Tivoli Directory Integrator 7.1.1 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 '*' is output instead of the actual value.

When connection parameters are found in the TaskCallBlock (TCB), the values never be logged directly by Tivoli Directory Integrator. The fact that parameters were given is 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 cryptoutils (see "The TDI Encryption utility" on page 126). Note that the TDI server only decrypts script files that have the ".jse" filename extension. The ".jse" extension indicates to the TDI server that the script file is encrypted. That is why, after you encrypt a Server Hook script file, make sure to change its filename extension to ".jse".

Remote Configuration Editor and SSL

The Configuration Editor used to edit remote Config Files (that is, Config files on a remote system) is called the Remote Configuration Editor (Remote CE). The Tivoli Directory Integrator 7.1.1 Remote CE is capable of starting AssemblyLines in configurations opened for editing. The Remote Configuration Editor is a client of the Server API of the remote TDI Server. Consequently the Remote Configuration 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 Configuration Editor connects must be configured to require SSL client authentication. This is a configuration of the Server API – for details see "SSL-based authentication" on page 107.
2. The Remote Configuration Editor TDI instance must be configured to supply SSL client authentication. This is configured in a "SSL client authentication" on page 96.

This SSL client authentication is needed because the Remote Configuration Editor uses listener objects so that it can be notified when an AssemblyLine 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 Configuration Editor

Using a Remote Configuration Editor is a little different from using a local CE. When running a remote Configuration Editor to manage a Config on a remote system, you must 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 must have Admin privileges on the remote Config Instance.
- When connecting to a data source (using **Connect** buttons in mapping windows), these connections are evaluated locally.

For example: `ldap://localhost:389` results in the Configuration Editor (CE) attempting to connect to the local LDAP server, rather than to the LDAP server on the remote computer.

- When generating WebServices-related connectors results in function components that generate the WSDL file, .jar files (using Complex Type Generator), and so on, you are generating them locally. These components are not generated on the remote system to which the CE is connected and must be uploaded manually 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.
- When working with **System Store** operations (such as deleting the Iterator state key, and so on) that are available in the CE, work with the local system store and not with the remote Tivoli Directory Integrator computer'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.
- When you use the **Parameter Substitution** editor (available with Ctrl-E), the editor shows only the local properties, and not the properties set on the remote system. Similarly, creating and saving a new property store (file type) stores the property store (file) locally.
- When using the Configuration Editor to edit remote Config files, you are subject to Server API authentication and authorization, because the CE is acting as a client application. Therefore, in order to use the CE in this way, you must have *admin* access on the Remote Server.
- When using the Remote Server, the Remote Server itself must have sufficient access to the local file system where the Config files are stored. If the ConfigFiles 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, you cannot edit remote Configs.

Summary of configuration files and properties dealing with security

Table 18. The table of configuration files that were discussed above and what is contained in each.

Configuration file	Location	Description
global.properties	<code>TDI_home/etc</code>	This file is the primary configuration file for the server.
solution.properties	Solution folder	This file (<code>solution.properties</code>) is initially a copy of <code>global.properties</code> used by the current solution. After you make changes, values in this file override corresponding values in <code>global.properties</code> .
registry.txt	<code>TDI_home/serverapi</code>	This file is the User registry for the Server API, defined by the "api.user.registry" property in <code>global.properties</code>
build.properties	<code>TDI_home/etc</code>	This file contains the Tivoli Directory Integrator build information, build date, version, and so on; it is a text file, and by default the file is in the platform-native encoding.

Table 18. The table of configuration files that were discussed above and what is contained in each. (continued)

Configuration file	Location	Description
tdisrvctl-log-4j.properties	<i>TDI_home/etc</i>	This file controls the logging strategy for the tdisrvctl command line utility.
Log4J.properties	<i>TDI_home/etc</i>	This file controls the logging strategy for the server (ibmdisrv) when started from the command line.
jlog.properties	<i>TDI_home/etc</i>	This file controls the tracing and First Failure Data Capture (FFDC) strategy
ibmdi.ico	<i>TDI_home/etc</i>	This file lists the icons for Tivoli Directory Integrator.
idisrv.sth	<i>TDI_home</i>	This file contains the Tivoli Directory Integrator server stash; it is a binary file that contains the encrypted password for the sample server keystore file (testserver.jks).
derby.properties	<i>TDI_home/etc</i>	This file contains the default configuration for the Derby System Store shipped with Tivoli Directory Integrator.
reconnect.rules	<i>TDI_home/etc</i>	This file contains text that defines reconnect rules for how Tivoli Directory Integrator should handle reconnect exceptions.
global.properties.v611	<i>TDI_home/etc</i>	This file serves as a sample place holder and is useful during migration.
TDI0701.SYS2	<i>TDI_home/etc</i>	This is the product signature (license) file used by the ITLM agent to recognize Tivoli Directory Integrator.
pkcs11.cfg	<i>TDI_home/etc</i>	This file is used for initializing the IBM PKCS11 implementation provider. For details refer to section PKCS11 Configuration File.
testadmin.der	<i>TDI_home/serverapi</i>	This file is the exported certificate from testadmin.jks.
testadmin.jks	<i>TDI_home/serverapi</i>	This file contains an example keystore and truststore for a Server API remote client.
cryptoutils.bat(sh)	<i>TDI_home/serverapi</i>	This file is a command line utility (shell script) used for encrypting and decrypting Tivoli Directory Integrator configurations and the user registry file.
testserver.jks	<i>TDI_home</i>	This file is a sample server keystore and truststore, referenced as an example.
testserver.der	<i>TDI_home</i>	This file is an exported sample server certificate, ready to be imported in a truststore.
am_config.properties	<i>TDI_home/ActionManager</i>	This file configures the Action Manager.
am_logging.properties	<i>TDI_home/ActionManager</i>	This file configures Action Manager logging.
ibmdiservice.props	<i>TDI_home/win32_service</i>	This file configures the Windows service.
mqeconfig.props	<i>TDI_home/jars/plugins/</i>	This file allows configuration of the MQe service. In Tivoli Directory Integrator, you can access MQe using authentication for the MQe Mini-Certificate Server to issue certificates; the certificates are then used for authentication. When authenticating, additional properties available in Tivoli Directory Integrator that must be added to the mqeconfig.props properties file.

Note: The file registry.txt can be encrypted and decrypted using the “The TDI Encryption utility” on page 126. The cryptoutil tool should not be applied on global.properties or solution.properties. You can encrypt individual property values but not the whole properties file.

Table 19. 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	On or off switch for secure mode.
api.keystore	file name	Server keystore used for SSL certificates. Previously com.ibm.di.server.keystore.
api.key.alias	Key alias	Key alias from keystore for SSL certificates. Previously com.ibm.di.server.key.alias.
{protect}-api.keystore.password	SSL keystore password	Keystore password for SSL. Added in TDI 7.0.
{protect}-api.key.password	SSL key password	Key password for SSL. Added in TDI 7.0.
com.ibm.di.server.encryption.keystore	file name	Data encryption for the keystore that hosts the key used by the Server. Added in TDI 7.0.
com.ibm.di.server.encryption.key.alias	Key alias	Encryption keystore key alias. Added in TDI 7.0
com.ibm.di.server.encryption.keystoretype	Keystore type, that is, "JKS", "JCEKS", and so on.	Keystore type that hosts the encryption key of the Server. Added in TDI 7.0.
com.ibm.di.server.encryption.transformation	"RSA" or some secret key transformation	Server transformation used for encryption. Can be set to either "RSA" (public key encryption) or to some secret key transformation (124 of the Tivoli Directory IntegratorServer Security section). Added in TDI 7.0.
api.on	true/false	On or off Server API switch.
api.user.registry	file name	Server API users registry file
api.user.registry.encryption.on	true/false	User registry switch for encrypted or not encrypted.
api.remote.on	true/false	On or off switch for remote Server API. The default setting is true.
api.remote.ssl.on	true/false	On or off switch requiring, or not requiring, SSL for the remote Server API.
api.remote.ssl.client.auth.on	true/false	On or off switch requiring, or not requiring, SSL client authentication for the remote Server API
api.truststore	file name	Server truststore.
api.truststore.pass	*	Trustore password.
api.remote.nonssl.hosts		Non-SSL addresses for accepting non-SSL IP connections.

Table 19. 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
api.custom.method.invoke.on	true/false	Server API methods for custom method invocation =true when allowed to be used, and =false when disallowed.
api.custom.method.invoke.allowed.classes		Server API classes that can be directly invoked by the Server API methods for custommethod invocation.
api.custom.authentication	Script file name or "[ldap]/[jaas]" for built in LDAP or JAASAuthentication	Custom authentication method.
api.custom.authentication.ldap.*		LDAP authentication configuration set of properties.
javax.net.ssl.*		Standard JSSE set of properties for keystore, truststore and their passwords
com.ibm.di.server.pkcs11	false	pkcs11 compliant crypto devices for SSL, required or not required. Added in TDI 7.0
{protect}-com.ibm.di.server.pkcs11.pass	administrator	Access password for pkcs11 compliant crypto device. Added in TDI 7.0
com.ibm.di.server.pkcs11.accl	false	Hardware cryptographic devices to be used for encryption when this property is set to true.

Note: All properties listed in the above table can be set in the configuration file `global.properties`, and can be protected by encryption using the {protect}- prefix (see section ““Standard TDI encryption of `global.properties` or `solution.properties`” on page 125” for details).

Web Admin Console Security

See “AMC and Action Manager security” on page 225.

Miscellaneous security aspects

HTTP Basic Authentication

Some Tivoli Directory Integrator 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 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 Tivoli Directory Integrator truststore and keystore (see section "Client SSL configuration of TDI components" on page 96) 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 (that is, the *ibmdisrv* or *ibmditk* shell scripts which start the TDI server and TDI Configuration Editor respectively). You 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 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 *SSLocket* or the *SSLSocket* 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 -keystorepass mykeystorepass
```

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 "mykeystorepass". The created certificate can then be configured for use by setting the following properties in the *global.properties* or *solution.properties* file:

```
api.key.alias=MyServerCertAlias
api.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 installation 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 or application submitting a change password request to the MQe Password Store Connector.

For more information on configuring MQe authentication with Mini-Certificates, see the "Authenticated MQe Access" section in Chapter 8 "MQ Everyplace Password Store" of the *IBM Tivoli Directory Integrator V7.1.1 Password Synchronization Plug-ins Guide*.

Chapter 7. Reconnect Rule Engine

Introduction

The Tivoli Directory Integrator Server supports reconnect rules that apply to certain error situations during the life of a Connector. The server takes measures, laid out in rules, based on conditions occurring when communicating with target systems.

The AssemblyLine polls the Reconnect Rule Engine every time a Connector raises an exception and the engine recommends a course of action for the current situation. The AssemblyLine code then acts in the proposed way. The possible actions to attempt are:

- to reconnect
- to leave the exception unhandled and let further error mechanisms like error hooks process it.

The *reconnect action* leads to a reconnect attempt only if reconnect is enabled by means of the options available in the Connector's "Connection Errors" tab in the CE. If reconnect is not enabled in this configuration, reconnect is not attempted in case of error regardless of the decision of the Reconnect Rule Engine.

Reconnecting basically involves automatic restart of the Connector and bringing it to its previous position (if so configured). This is done by executing terminate for the Connector, then executing initialize for the Connector and in case of Iterator Connectors, optionally skipping entries until the position before the reconnect is reached. On each reconnect attempt the corresponding reconnect hook is invoked. The script in the hook may eventually change the configuration so that a subsequent reconnect would be successful. If the user has specified failover, an automatic failover or fallback is attempted when the reconnect attempts fail.

The *error action* implies that no automatic reconnect is attempted and that the corresponding error hooks are invoked. The hooks can eventually perform some custom recovery or error reporting.

Reconnect Rules

The Reconnect Rule Engine makes decisions based on configured rules. Each rule describes what should be done when a given kind of error situation ensues. The engine uses two types of rules:

- **Built-in rules**, which are stored in the tdi.xml files of each connector file and are packaged in the connector's jar file; as a result these rules are always specific to the particular connector class and match all connector names; this list of rules is the default list of the Reconnect Rule Engine when working on an error situation for a given Connector; if you have programmed your own Connectors in Java, then for information about how to construct your own built-in rules see section "Connector Reconnect Rules definition" in the "Implementing your own Components in Java" appendix in *IBM Tivoli Directory Integrator V7.1.1 Reference Guide*.
- For compatibility with previous releases of TDI, when the Reconnect Rule Engine is set up it implicitly adds to the built-in rules, a set of rules that prescribe to attempt reconnect on all IOException-s and all CommunicationException-s (java.io.IOException and javax.naming.CommunicationException);
- **User-defined rules**, which are loaded from an external text file named etc/reconnect.rules; this list of rules overrides the built-in rules. See "User-defined rules configuration" on page 139.

Each rule applies to certain connectors and certain error situations.

A rule has the following parts:

- **Connector Class**: the Java Class of the connectors to which the rule applies

- **Connector Name:** the name of the connector component as it is specified in the configuration file of the currently executed solution
- **Exception Class:** the base class of the exceptions to which the rule applies
- **Regular Expression:** a regular expression that matches the messages of the exceptions to which the rule applies
- **Action:** the action, prescribed by the rule. Can be *error* or *reconnect*.

An error situation is described by the following parts:

- **Connector Class:** the class of the connector that raised the exception
- **Connector Name:** the name of the connector that raised the exception
- **Exception:** the exception raised by the connector – a subclass of `java.lang.Throwable`.

A rule applies to an error situation if all of the following conditions are fulfilled at the same time:

- the rule applies to the connector in the error situation (subclasses of the connector class, described in the rule are also matched)
- the rule applies to the name of the connector which caused the error situation
- the exception is an instance of the exception class, to which the rule applies
- the rule does not have a regular expression to match the exception message or the regular expression matches the message of the exception.

When a given error situation occurs, the reconnect rule engine finds the most specific rule that matches the error situation. First the engine searches through the user-defined rules and if no matching rule is found, it searches the built-in rules. If still no matching rule is found, the engine prescribes the default action, which is "error". If a matching rule is found in the user-defined rules, then the built-in rules are not searched, even if there exists a more specific rule in the built-in rules.

Note: If two or more rules match an error situation, the most specific rule is selected; if there are several most-specific rules and none of them is more specific than the rest, then the first rule in the list is selected. That is why the order of the rules in the rule lists matters. For example: suppose the following rules exist (this is pseudo-syntax used for clarity only):

```
...exceptionClass = "java.io.IOException", exceptionMessageRegExp = ".*", action = "error"...
...exceptionClass = "java.io.IOException", exceptionMessageRegExp = "\w*", action = "reconnect"...
```

If an exception of type `java.io.Exception` with message "problem" is raised, then the first rule is selected, although both rules match the error and no rule is more specific than the other (the outcome of the regular expression match is not considered for weighting purposes.)

Nested Exceptions:

Some exceptions are nested inside other exceptions. When the reconnect rule engine searches through a list of rules (for example the built-in rules), the engine searches for a rule that matches the top-level exception first. If no matching rule is found, then the engine searches again the same list of rules but this time it searches for a match for the nested exception (if the top-level exception has no nested exception this search is skipped). Note that only the first-level nested exception is attempted to be matched by the reconnect rule engine; if there are more levels of nested exceptions they are ignored.

Note: Automatic Failover is not possible for Server mode Connectors.

User-defined rules configuration

The list of user-defined rules is configured in a text file named reconnect.rules in the "etc" subfolder of the TDI solution folder (or the TDI installation folder, if no solution folder has been defined). Each rule is placed on a single line. The format of a rule is as follows:

```
<connector_class>:<connector_name>:<exception_class>:<action>:<regular_expression>
```

where

- <connector_class> is the fully qualified name of the Java class of the Connector
- <connector_name> is the name of the Connector as inserted in the AssemblyLine
- <exception_class> is the fully qualified name of the Java class of the exception
- <action> can be either 'error' or 'reconnect'
- <regular_expression> is a Java regular expression as described in the JavaDoc of the java.util.regex.Pattern class at <http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html>.

Notes:

1. Each part except the action can be empty. If a part is empty that means "match-all".
2. Each part is mandatory – even if it is empty the surrounding colons must be present. (Consequently on each line there must be at least 4 colons – each colon separating two adjacent parts of the rule. At least 4, because the regular expression may contain colons too. These colons do not interfere with the rule parsing because the regular expression comes last in a rule.)
3. No redundant white space is allowed.
4. The regular expression starts just after the fourth colon and spans until the end of the line.
5. The user-defined rules file is not a Java properties file. The main reason is that a key for a rule must include all rule parts, except the reconnect action, in order to be unique. So the only value from using the Java properties mechanism would be the separation of the action from the other rule parts. However, it would come at the price of escaping white-spaces, colons and equal signs (requirements for a valid property key). Even if the Java property framework was used, custom parsing of the property key would still be required in order to extract the rule parts from it.
6. The regular expression (not the reconnect action) comes last on each line. This pattern is chosen such that it is unnecessary to escape colons (which are considered rule part delimiters) in the regular expression.
7. The regular expression must match the entire message text: Suppose the message text you want to match contains the words "Some Error" somewhere in the message text. A suitable regular expression might then be:

```
.*Some Error.*
```

The character "." matches any character except new line, and the * modifier specifies 0 or more. Now suppose the message ends with a new line. If that is the case, the previous regular expression does not match. You can try a regular expression like this instead:

```
.*Some Error.*\r?\n?
```

"\r" and "\n" specify return and new line characters, and the ? modifier specifies 0 or 1 occurrence.

8. You must still configure reconnect in the Connector's configuration; see "General reconnect configuration" on page 140.

Examples

An example, consisting of two rules:

```
com.ibm.di.connector.ReconnectTestConnector:myconnname:java.io.IOException:error:.*\Wfatal\W.*  
::java.io.IOException:reconnect:
```

Reconnect with the JDBC Connector:

Tivoli Directory Integrator's JDBC connector is configured in Iterator mode to iterate a table from DB2 and is enabled for the reconnect feature. However, at the time of running the solution, DB2 instance is not started yet. In order to have reconnect working, the following exception details need to be mentioned in the reconnect.rules file:

```
com.ibm.di.connector.JDBCConnector::com.ibm.db2.jdbc.DB2Exception:reconnect:
```

Reconnect with the RAC Connector:

Tivoli Directory Integrator's RAC connector is configured in Iterator mode and is enabled for the reconnect feature. In case the Agent Controller server is down, in order for the RAC connector to try to reattempt (reconnect), the following exception details need to be mentioned in the reconnect.rules file:

```
com.ibm.di.connector.RACConnector::org.eclipse.tptp.platform.execution.exceptions.AgentControllerUnavailableException:reconnect:
```

Exception considerations

Every environment and solution created for a particular environment using Tivoli Directory Integrator is typically unique. User-defined rules are custom-built and the functionality is made available so solutions can automatically attempt to reconnect based on the exceptions specific to the environment or solution. Refer to the Tivoli Directory Integrator Java API documentation for information about specific exceptions that are returned by the Tivoli Directory Integrator APIs for each Connector.

Additionally, some Tivoli Directory Integrator components rely on underlying libraries and the APIs of these libraries throw exceptions for specific situations. Below we list a few core TDI components where you can look for additional information on exceptions and what may be the cause of the exceptions. This information is helpful when deciding if you want to attempt to create custom reconnect rules for specific exceptions that may be encountered:

- LDAP Connector - The LDAP Connector depends on the JNDI libraries shipped with the JRE. For more information on the JNDI interface, its APIs, and the exceptions it may throw, see <http://java.sun.com/j2se/1.5.0/docs/api/javax/naming/package-summary.html>.
- JDBC Connector - The JDBC Connector depends on the configured JDBC Driver. The Java API documentation or reference material for the configured JDBC driver should be consulted for more information on the possible exceptions that may be thrown. The "Understanding JDBC Drivers" subsection in the JDBC Connector chapter in *IBM Tivoli Directory Integrator V7.1.1 Reference Guide* contains links to the JDBC Driver documentation for a set of commonly used JDBC drivers.

General reconnect configuration

Specifying a reconnect rule is necessary for a reconnect to be attempted. However it is not sufficient by itself. The other requirement is enabling reconnect in the general reconnect configuration. This can be done under the **Connection Errors** tab in the Configuration Editor . If reconnect is not enabled in this configuration, reconnect is not attempted in case of error regardless of the decision of the Reconnect Rule Engine. Here is a list of Configuration options:

Number Of Retries

The number of times a reconnect attempt is made when a problem occurs, before giving up. If a new problem occurs later on, the same number of attempts is made.

Delay Between Retries

The number of seconds to wait between each reconnect attempt, and before the first reconnect attempt.

Retry Connect on Initial Connection Failure

If this flag is set, and a connection cannot be established when the connector is being initialized, a "reconnect" attempt is made. Not really reconnect, since a connection was not established in the first place, but generally the same mechanism.

Auto Reconnect on Connection Loss

If this flag is set, and the connection is lost after the connector is initialized, a reconnect attempt is made.

Auto Skip Forward

After a reconnect, automatically skip forward as many times as the number of successful reads.

Automatic Failover

If this flag is set, an automatic failover is attempted after an automatic reconnect fails.

FailOver Connector

Name of the Connector in the Resources Library that is used for automatic failover.

Fallback After

If this field is set to a positive value, after the specified seconds have passed, an automatic fallback is attempted. If the fallback fails, it is not attempted again before the specified seconds have passed again.

Note: For both the **Retry Connect on Initial Connection Failure** and **Auto Reconnect on Connection Loss** flags, the reconnect engine is determine if the exception leads to a reconnect attempt, or is a more general error.

Chapter 8. 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 use 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 V7.1.1 Reference Guide*) does not talk directly to the System Queue, but rather uses the Server API as an intermediary.

In IBM Tivoli Directory Integrator 7.1.1, the System Queue is enabled by default by the install process.

System Queue Configuration

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

systemqueue.on

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

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 four standard Tivoli Directory Integrator 7.1.1 JMS Driver implementations:

- `com.ibm.di.systemqueue.driver.ActiveMQ` (“Apache ActiveMQ parameters,” default JMS Provider)
- `com.ibm.di.systemqueue.driver.IBMQe` (“WebSphere MQe parameters” on page 145)
- `com.ibm.di.systemqueue.driver.IBMQ` (“WebSphere MQ parameters” on page 145)
- `com.ibm.di.systemqueue.driver.IBMMB` (“Microbroker parameters” on page 146)
- `com.ibm.di.systemqueue.driver.JMSScriptDriver` (other JMS system by way of the “JMSScript Driver parameters” on page 146)

The default value is `com.ibm.di.systemqueue.driver.ActiveMQ`.

Depending on the `systemqueue.jmsdriver.name` parameter, one of the following sections is applicable:

Apache ActiveMQ parameters

To use ActiveMQ as the JMS provider for the System Queue, set the `systemqueue.jmsdriver.name` property in `global.properties/solution.properties` to `com.ibm.di.systemqueue.driver.ActiveMQ`. The ActiveMQ driver has the following parameter.

- **jms.broker** - the ActiveMQ server address (Protocol, IP address, and TCP port number). For example, `tcp://localhost:6161` or `ssl://localhost:616171` to use SSL connection.

The default value is:

`vm://localhost?brokerConfig=xbean:etc/activemq.xml`

This value runs the ActiveMQ in embedded mode. The `etc/activemq.xml` file holds the default ActiveMQ configuration.

Notes:

1. The path to the ActiveMQ configuration XML file (after xbean:) cannot hold spaces. For more information, see <https://issues.apache.org/activemq/browse/AMQ-1385>.
If the path contains spaces, each space character must be URL encoded three times, thus transforming it to %2520.
2. The System Queue initializes ActiveMQ at startup if the **systemqueue.on=true** parameter is set to true in the `solution.properties` file.

Configuration

ActiveMQ configuration relies on the `activemq.xml` file, located at `TDI_install_folder/etc`. The ActiveMQ configuration parameters are as follows.

broker

This ActiveMQ message broker consists of transport connectors, network connectors and properties that are used to configure the broker. The attributes are:

- `brokerName="localhost"` - the name of the broker.
- `dataDirectory=".//ActivemqDataStore"` - the directory, which is used to store the data of ActiveMQ.
- `useShutdownHook="true"` - sets whether or not a shutdown handler is used to close the broker if the JVM is terminated.
- `useJmx="true"` - sets whether or not the services of the broker to be exposed into JMX.

managementContext

This parameter configures how the ActiveMQ is exposed in JMX. The attributes are:

- `createConnector="true"` - sets whether or not the ActiveMQ creates its own JMX connector.
- `o connectorPort="1099"` - the port of the Connector. The value is 1099 by default.

persistenceAdapter/kahaDB

This parameter configures message persistence for the broker. The attributes are:

- `journalMaxFileLength="32mb"` - sets the maximum size of the message data logs.
- `checksumJournalFiles="true"` - creates a checksum for a journal file to enable checking for the corrupted journals.
- `checkForCorruptJournalFiles="true"` - if enabled, checks for corrupted journal files on startup and try and recover them.

transportConnectors

This parameter consists of transport connectors that the ActiveMQ listens to. The attributes are:

- `name="openwire"` - the name of the transport connector.
- `uri="tcp://localhost:61616"` - the address of the transport connector.

Note: For more information about the XML objects used in the XML configuration file, refer to the ActiveMQ's XBean XML Reference 5.0 at <http://activemq.apache.org/xbean-xml-reference-50.html>.

Logging

The ActiveMQ relies on log4j to log information in the broker client and the broker. The following lines inside `log4j.properties` configure the ActiveMQ logging by setting the default logging categories of ActiveMQ items:

- `log4j.logger.org.apache.activemq=INFO`
- `log4j.logger.org.apache.activemq.spring=WARN`
- `log4j.logger.org.apache.activemq.web.handler=WARN`
- `log4j.logger.org.springframework=WARN`
- `log4j.logger.org.apache.xbean=WARN`

- log4j.logger.org.apache.camel=ERROR

Using SSL with ActiveMQ

You can configure the ActiveMQ to use SSL connection by specifying the `<sslContext>` element and the correct transportConnector's URI in the XML configuration file of ActiveMQ.

ActiveMQ relies on certificates to use SSL connection. By default ActiveMQ is configured to reuse the TDI Server API certificates located at the `TDI_install_folder/serverapi` folder as `keyStore` and `trustStore`. To reuse, the names of the client and server keystore files must be specified in the `<sslContext>` element of the configuration file of ActiveMQ. For example:

```
<sslContext>
  <sslContext
    keyStore="file:./serverapi/testadmin.jks" keyStorePassword="administrator"
    trustStore="file:./serverapi/testadmin.jks" trustStorePassword="administrator"/>
</sslContext>
```

Where, `testadmin.jks` is the name of the TDI certificate and `password` is the password of the TDI certificate.

Note: The `<sslContext>` element and all the parameters are taken into account only when the `javax.net.ssl` properties are not set in the TDI `solution.properties` file. By default, the ActiveMQ reuses the javax properties from the TDI API instead of the properties set in the `<sslContext>` tag.

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 149 bundled with Tivoli Directory Integrator

systemqueue.jmsdriver.param.mqe.file.ini

This is an MQe-specific parameter that specifies the relative 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 `MQePWStore/pwstore_server.ini`. This is the default location for the MQe initialization file created by the “MQe Configuration Utility” on page 149.

The system queue is turned on by default, except on z/OS. If you want to use MQe as a system queue you then an abridged enabling procedure is as follows:

1. Set the `systemqueue.on` property in the `global.properties` or `solution.properties` file to *true*.
2. Configure MQe by invoking:

```
cd solution_dir (if using the installation directory, use cd TDI_install_dir)
TDI_install_dir/jars/plugins/mqeconfig.sh
TDI_install_dir/jars/plugins/mqeconfig.props create server (one line)
```

WebSphere MQ parameters

These are WebSphere MQ-specific parameters; for more information about these parameters, see the MQ JMS diver initialization properties in the “System Queue Configuration Example” on page 148 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, for example, SSL_RSA_WITH_RC4128_MD5)

systemqueue.jmsdriver.param.jms.sslUseFlag

(true for SSL connection requested, false if not)

Microbroker parameters

In order to use Microbroker (MB) as the JMS provider for the System Queue, the `systemqueue.jmsdriver.name` property in `global.properties` or `solution.properties` must be set to `com.ibm.di.systemqueue.driver.IBMMB`.

The Microbroker driver has the following parameters (listed here without the "systemqueue.jmsdriver.param." prefix):

jms.broker

the MB server address (IP address and TCP port number); an example value would be "9.126.6.120:1883"

jms.clientID

the client ID; it is required.

Note: In order to be able to use Microbroker as the JMS provider for the System Queue, some Microbroker jars are needed. A sample list of the required jars is available in section External System Configuration, Microbroker of the JMS Connector in *IBM Tivoli Directory Integrator V7.1.1 Reference Guide*.

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 (that is, taken into account when the `systemqueue.jmsdriver.name` is set to `com.ibm.di.systemqueue.driver.JMSScriptDriver`) that 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, see the JMS driver settings in the "System Queue Configuration Example" on page 148 section. 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 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` or `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` or `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 use the third-party Fiorano MQ system is provided in the `TDI_install_dir/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` or `solution.properties`. Using the `env` object for parameter retrieval would make changing the configuration easier, because only properties in `global.properties` or `solution.properties` would have 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.IBMQM  
systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.ActiveMQ  
  
### 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=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  
  
### ActiveMQ driver initialization properties  
## Specifies the location of the ActiveMQ initialization file.  
## This file is used to initialize ActiveMQ on TDI server startup.  
systemqueue.jmsdriver.param.jms.broker=vm://localhost?brokerConfig=xbean:etc/activemq.xml  
  
## 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, see the "System Queue Configuration" on page 143 section.

MQe Configuration Utility

To configure and use MQe as the default system queue, set up MQ Queue Manager in the new Solution Directory using MQe Configuration Utility. The MQe Queue Manger setup has two predefined queues:

- **_default** – serves as a general purposes queue
- **passwords** – serves as a general purposes queue passwords. This queue is used by the JMS Password Store components for storage of password changes. This makes the System Queue more usable.

The Tivoli Directory Integrator 7.1.1 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.

Additionally the Tivoli Directory Integrator 7.1.1 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 creates 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 delete the MQe Queue named "queue_name" using the `mqeconfig.props` configuration file:

```
mqeconfig mqeconfig.props delete queue queue_name
```

If your solution needs any special configuration, then you can use the MQe Explorer to fine tune your MQe configuration. The MQe Explorer is not bundled with Tivoli Directory Integrator, but can be downloaded as part of the MQe Server Support ES06 pack at http://www-1.ibm.com/support/docview.wss?rs=0&dc=D400&q1=MQe&q2=MQ+Everyplace&uid=swg24007943&loc=en_US&cs=utf-8&cc=us&lang=en.

Authentication of MQe messages to provide MQe Queue Security

In Tivoli Directory Integrator 7.1.1 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 Tivoli Directory Integrator 7.1.1 must be added to the `mqeconfig.props` properties file, which contains the configuration properties of the MQe Configuration Utility.

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 include an option to renew certificates. Typing the following command renews the certificates:

```
mqeconfig mqeconfig.props renewcert {client | server}
```

- When the last command option is "client", the following values must 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 computer 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.
- When the last command option is "server", the following values must 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 computer 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 plug-ins have been configured for multiple Windows Domain Controllers—in this case, it is likely that there separate instances of MQe client side Queue Managers with the name "PWStoreClient". Additionally, for each of the client Queue Managers, there is 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 (that is, "PWStoreClient", "PWStoreServer+passwords") is 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 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 is configured using the mqeconfig utility. When this utility executes it 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 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 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 is created. If you provide these two parameters, a Connection definition 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 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 9. Encryption and FIPS mode

To provide confidentiality of data, IBM Tivoli Directory Integrator 7.1.1 can encrypt:

- configuration files
- property values in properties files
- server user registry files
- JavaScript files

Encryption is the process of selecting some humanly readable text, called *plaintext*, and hiding its content and meaning to make the data in the plaintext format more secure. Plaintext is written in lowercase letters. Encrypted text is called *ciphertext*. Ciphertext is written in capital letters.

Note: In Config files, if the {protect}- prefix precedes the name of a property, then the property value is, or should be, encrypted. The prefix, {protect}- is optional. The values that are already encrypted values start with {encr}.

See “Working with encrypted TDI configuration files” on page 123 and “Encryption of properties in external property files” on page 125.

For example:

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

The {java} value must be b64-encoded. For example:

```
{protect}-api.truststore.pass={encr}J8AKimpEtu3Bb10Vg55F/5d5v02kXWcNUWnCq3vINUc6K0719z9dEk3H430t2iTT1dZTI6FSSV  
in9KsCyBLmgv+n84w7He1K13ro2dFmZbTYKMXuxGoqN9nL2V0vZoptNqzoWvs6IN/p3VkJIBt1ao/9mEPEKuIwRnKtkQ89Bg=
```

Configuring Tivoli Directory Integrator to run FIPS mode

The Federal Information Processing Standard (FIPS) Publication 140-2, FIPS PUB 140-2, is a U.S. government computer security standard used to accredit cryptographic modules.

When the Tivoli Directory Integrator server is configured to run in FIPS mode, that Server is using the FIPS 140-2 certified cryptographic modules. Tivoli Directory Integrator does not generate cryptographic keys – keys are created using external utilities such as *keytool* and *Ikeyman*). For information on Tivoli Directory Integrator use of encryption, see Chapter 6, “Security and TDI,” on page 89. 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 executable file, *keytool.exe* is found in *root_directory/jvm/jre/bin*, or *root_directory/jvm/bin*, depending on your platform.

Symmetric cipher support

Symmetric cipher support in Tivoli Directory Integrator is one of the requirements for achieving a FIPS 140-2 compliance.

The reason for encrypting a message is to change the message into a meaningless form of text called cipher text that is meaningless to whoever intercepts the message. There are many different encryption algorithms called ciphers. One of the most widely known ciphers is the *symmetric* cipher. The symmetric cipher has a key that both the sender and the receiver can keep. The sender uses that key to encrypt the message. The receiver uses the same key to decrypt the message.

An optional configuration is provided to use a symmetric cipher (specifically, the Advanced Encryption Standard, or AES). The symmetric cipher encoded using AES allows customers that need FIPS-compliant solutions to use a supported cipher around encryption.

The following property defines the cipher:

```
com.ibm.di.securityTransformation=DES/ECB/NoPadding
```

This property defines a cipher for the password-based encryption or decryption of Tivoli Directory Integrator configurations.

FIPS encryption

You can run Tivoli Directory Integrator and the Tivoli Directory Integrator server in a secure way using FIPS. You can also configure additional properties when you want to operate Tivoli Directory Integrator in a specific mode, for example, FIPS mode.

Connectors, Function Components, Parsers: FIPS 140-2 is concerned only with cryptographic functionality such as SSL, digital signing, encryption, cryptographic hashing and random number generation.

SSL:

FIPS 140-2 requires TLS to be the protocol for SSL communication. SSLv3 and its predecessors are not allowed. When FIPS mode is turned on, Tivoli Directory Integrator components that use SSL will fail to communicate with external systems that do not support TLS.

JDBC and the System Store:

The DB2 Type 4 JDBC driver (`com.ibm.db2.jcc.DB2Driver`) that is shipped with Tivoli Directory Integrator, supports SSL in a FIPS conformant way.

The Apache Derby drivers, network and embedded, do not support SSL in version v10.5.3 (which is the one bundled with Tivoli Directory Integrator).

However, the Apache Derby v10.5.3 database engine can perform database encryption. By default Tivoli Directory Integrator uses Derby for its System Store. If you use database encryption functionality of Derby in FIPS mode, be sure to specify the IBM certified cryptographic provider IBMJCEFIPS as the provider used for encryption and also choose a FIPS approved encryption cipher. Here is an example of how to configure the System Store to use Derby with FIPS compliant database encryption:

```
com.ibm.di.store.database=jdbc:derby://localhost:1527/C:\TDI\TDISysStoreEnc;create=true;
  dataEncryption=true;encryptionKey=c566bab9ee8b62a5ddb4d9229224c678;encryptionAlgorithm=AES/CBC/NoPadding;
  encryptionProvider=com.ibm.crypto.fips.provider.IBMJCEFIPS
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
```

JMS and the System Queue:

MQe Mini-Certificates involve cryptography that is not FIPS compliant, so this security feature of MQe should not be used in FIPS mode.

The WebSphere MQ 5.3 JMS provider is not capable of running SSL in a FIPS compliant mode. In FIPS mode SSL should not be used with that provider.

The WebSphere MQ 7.1 JMS provider can use SSL in a FIPS compliant mode. To take advantage of it, however, you need MQ 6.0.1.0 or higher, because in earlier versions of MQ 7.1 the FIPS conformant mode does not work properly with Java 6.0 that Tivoli Directory Integrator uses. To use FIPS compliant SSL communications between Tivoli Directory Integrator and WebSphere MQ:

1. Ensure that the WebSphere MQ installation is of version 6.0.1.0 or higher.
2. Ensure that the corresponding Queue Manager on the MQ side requires FIPS compliant SSL communications.

3. Ensure that the corresponding SSL channel of the Queue Manager uses a FIPS compliant SSL Cipher Spec.
4. Turn on FIPS mode for Tivoli Directory Integrator. When FIPS mode is enabled for Tivoli Directory Integrator, it automatically enables FIPS mode on all JMS SSL connections to WebSphere MQ.
5. Copy the JMS client jars from the WebSphere MQ installation to TDI; refer to the JMS Connector documentation in *IBM Tivoli Directory Integrator V7.1.1 Reference Guide* for a list of necessary client libraries for MQ 7.1 and how to deploy them in TDI.
6. On the Tivoli Directory Integrator side, configure a FIPS compliant SSL Cipher Suite that is compatible with the SSL Cipher Spec configured on the SSL channel of the MQ Queue Manager. You can do this using the **jms.sslCipher** parameter of the JMS Connector and the **systemqueue.jmsdriver.param.jms.sslCipher** system property of the MQ driver for the System Queue. You can find a SSL Cipher Spec to Cipher Suite mapping and their FIPS compliance here: http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzaw.doc/uj34740_.htm.

The TDI server and FIPS: When run in this mode the Tivoli Directory Integrator Server is forced to use FIPS 140-2 cryptographic modules

Note: If the Server is running with FIPS and SSL enabled, then do not use clients with SSL for secure sockets communication. In this case the Server uses TLS and a connection will not succeed. Instead of using SSL make sure you are using TLS like the Server does for secure sockets communication.

Running the Tivoli Directory Integrator Server in FIPS mode has the following implications:

- Only FIPS compliant crypto algorithms are allowed for encryption and decryption of configurations, properties, and so forth.
- Auxiliary tools which use encryption/decryption should be used in FIPS compliant way - Ikeyman, createstash, cryptoutils, keytool, and so forth.
- Components will not be able to communicate with external systems that do not use TLS for socket communication.
- Some components should not be used when the Server is in FIPS mode because they will break the FIPS compliancy. Refer to Table 20 on page 156 for a list detailing component compliance.

Enabling FIPS mode: When using FIPS, many Tivoli Directory Integrator configuration options are changed, so you must keep in mind several rules in order to maintain FIPS compliancy. Some of the rules are mentioned in this document and others can be found in <https://w3.webahead.ibm.com/w3ki/download/attachments/370821/FIPS+140+Guidelines.pdf?version=1> and <http://www.ibm.com/developerworks/java/jdk/security/60/FIPShowto.html>.

Enabling FIPS mode in TDI:

1. Set the **com.ibm.di.server.fipsmode.on** property to **true** in **global.properties** or **solution.properties**.
2. Make sure the **com.ibm.di.securityTransformation** property value is in an algorithm which is FIPS compliant, for example, AES/ECB/NoPadding. This algorithm is used when you attempt to open an encrypted configuration.
3. Hardware cryptography can not be used along with SSL in FIPS mode.
The underlying SSL module – IBMJSSE2 does not support hardware cryptography in FIPS mode as stated here: <http://www-128.ibm.com/developerworks/java/jdk/security/60/secguides/jsse2Docs/JSSE2RefGuide.html#runfips>. You cannot use hardware-based SSL keys for the Server API in FIPS mode; the **com.ibm.di.server.pkcs11** property must be absent or set to false in **global.properties** and **solution.properties**.
4. Make sure Server encryption uses a transformation that is FIPS 140-2 compliant.

By default the Server uses public key encryption with the RSA algorithm. However, the RSA encryption option is not compliant with FIPS 140-2. That is why you must manually configure another cryptographic transformation that is FIPS allowed. Below are sample steps that setup Tivoli Directory Integrator to use the AES cipher for encryption:

- Generate an AES secret key and put it in a keystore. This can be done using the keytool utility located in the bin folder of a Tivoli Directory Integrator installation like this:

```
keytool -genseckeys -alias server -keyalg AES -keysize 128 -keystore server.jck -storepass mypass -storetype jceks  
-keypass mykeypass -providerClass com.ibm.crypto.fips.provider.IBMJCEFIPS
```

This command creates a new keystore file server.jck of type JCEKS (JKS keystores cannot host secret keys) with an AES key of size 128 under alias *server*. The password for the created keystore is *mypass*. Pay special attention to the *keygenproviderclass* parameter – it is absolutely necessary to specify the FIPS certified provider if you strive for FIPS 140-2 compliance. Note that this is just an example, you can use whatever file names, passwords and aliases you wish.

- Change the Tivoli Directory Integrator settings to use secret key encryption with the newly generated key. For example in global.properties or solution.properties file, set the following properties:

```
com.ibm.di.server.encryption.keystore=server.jck  
com.ibm.di.server.encryption.storetype=jceks  
com.ibm.di.server.encryption.key.alias=server  
com.ibm.di.server.encryption.transformation=AES/CBC/PKCS5Padding
```

- Migrate all existing files that have been encrypted with the old key:

All encrypted files that existed prior to the introduction of the new key, need to be migrated. Migration involves decryption with the old key and (optionally) re-encryption with the new one (see “[Maintaining encryption artifacts – keys, certificates, keystores, encrypted files](#)” on page 164). For example you can migrate global.properties as follows:

```
cryptoutils -input ..;/etc/global.properties -output ..;/etc/global.properties  
-mode decrypt_props -keystore ..;/testserver.jks -storepass server -alias server  
-transformation RSA -storetype jks -keypass server

cryptoutils -input ..;/etc/global.properties -output ..;/etc/global.properties  
-mode encrypt_props -keystore ..;/server.jck -storepass mypass -alias server  
-transformation AES/CBC/PKCS5Padding -storetype jceks -keypass mykeypass
```

- Regenerate the Tivoli Directory Integrator Server “Stash File” on page 122 to reflect the new passwords of the encryption keystore and the encryption key. This is done using the createstash utility found in the bin folder of a Tivoli Directory Integrator installation. For example:

```
createtash mypass mykeypass com.ibm.crypto.fips.provider.IBMJCEFIPS
```

- Use only FIPS compatible Tivoli Directory Integrator components in your solutions, as listed in the table below:

Table 20. FIPS compatible components

Directory Integrator Component	Allowed in FIPS mode?	Remarks
Connectors		
Active Directory Change Detection Connector	yes	Uses default JSSE factories for SSL
AssemblyLine Connector	yes	Operates as a Server API client
Axis Easy Web Service Server Connector	yes	Uses default JSSE factories for SSL
Command line Connector	yes	Provides no cryptography features
Domino/Lotus Notes Connectors	no	Domino/Notes 7 cryptographic capabilities are not FIPS conformant. (Some FIPS enablement may be included in Notes 8.0.1.)

Table 20. FIPS compatible components (continued)

Directory Integrator Component	Allowed in FIPS mode?	Remarks
ITIM DSMLv2 Connector	yes	Uses default JSSE factories for SSL
DSMLv2 SOAP Connector	yes	Uses default JSSE factories for SSL
DSMLv2 SOAP Server Connector	yes	Uses default JSSE factories for SSL
Exchange Changelog Connector	yes	Uses default JSSE factories for SSL
File Connector	yes	Provides no cryptography features
FTP Client Connector	yes	Provides no cryptography features
GLA Connector	yes	Provides no cryptography features
HTTP Client Connector	yes	Uses default JSSE factories for SSL
Old HTTP Client Connector	yes	Uses default JSSE factories for SSL
HTTP Server Connector	yes	Uses default JSSE factories for SSL
Old HTTP Server Connector	yes	Provides no cryptography features
IBM Directory Server Changelog Connector	yes	Uses default JSSE factories for SSL
ITIM Agent Connector	yes	Provides no cryptography features
JDBC Connector	depends	If no cryptography features are used (SSL, encryption), the Connector is FIPS conformant. Otherwise FIPS conformance depends on the FIPS conformance of the cryptographic functionality of the JDBC driver that is used. See “Connectors, Function Components, Parsers” on page 154 for a discussion on the FIPS conformance of JDBC drivers.
JMS Connector	depends	If no cryptography features are used (SSL, encryption), the Connector is FIPS conformant. Otherwise FIPS conformance depends on the FIPS conformance of the cryptographic functionality of the JDBC driver that is used. See “Connectors, Function Components, Parsers” on page 154 for a discussion on the FIPS conformance of JMS providers.
JMX Connector	yes	Provides no cryptography features
JNDI Connector	yes	Uses default JSSE factories for SSL
LDAP Connector	yes	Uses default JSSE factories for SSL
LDAP Server Connector	yes	Uses default JSSE factories for SSL
Mailbox Connector	yes	Uses default JSSE factories for SSL
Memory Queue Connector	depends	Depends on the FIPS compliance of the JDBC driver used for the System Store. (The Memory Queue uses the System Store for persistence.) See “Connectors, Function Components, Parsers” on page 154 for a discussion on the FIPS conformance of JDBC drivers.
Memory Stream Connector	yes	Provides no cryptography features

Table 20. FIPS compatible components (continued)

Directory Integrator Component	Allowed in FIPS mode?	Remarks
MQe Password Store Connector	depends	Only PKCS#7 is allowed in FIPS mode for message protection. The RSA encryption option must not be used. The MQe Mini Certificates are not FIPS compliant, so they must not be used in FIPS mode.
Sun Directory Change Detection Connector	yes	Uses default JSSE factories for SSL
Properties Connector	depends	If encryption is turned off, the Connector is FIPS conformant. Otherwise FIPS conformance depends on the cipher used for encryption. An example of a FIPS 140-2 approved cipher is AES. Other approved ciphers can be found at: http://csrc.nist.gov/publications/fips/fips140-2/fips1402annexa.pdf The Server encryption option will always be FIPS conformant as long as TDI is configured correctly for FIPS mode. (See "Enabling FIPS mode" on page 155.)
Server Notifications Connector	yes	Operates as a Server API client
System Queue Connector	depends	If no cryptography features are used by the System Queue (SSL, encryption), the Connector is FIPS conformant. Otherwise FIPS conformance depends on the FIPS conformance of the JMS provider that is used by the System Queue. See "Connectors, Function Components, Parsers" on page 154 for a discussion on the FIPS conformance of JMS providers.
Windows Users and Groups Connector	yes	Provides no cryptography features
System Store Connector	depends	Depends on the FIPS compliance of the JDBC driver used by the System Store.
RAC Connector	yes	Provides no cryptography features
RDBMS Changelog Connector	depends	Same as the JDBC Connector
SNMP Connector	yes	Provides no cryptography features
SNMP Server Connector	yes	Provides no cryptography features
TAM Connector	yes	Tivoli Access Manager Runtime for Java is FIPS conformant
TCP Connector	yes	Uses default JSSE factories for SSL
TCP Server Connector	yes	Uses default JSSE factories for SSL
Timer Connector	yes	Provides no cryptography features
URL Connector	yes	Provides no cryptography features
Web Service Receiver Server Connector	yes	Uses default JSSE factories for SSL
z/OS Changelog Connector	yes	Uses default JSSE factories for SSL
Function Components		
Castor Java to XML FC	yes	Provides no cryptography features

Table 20. FIPS compatible components (continued)

Directory Integrator Component	Allowed in FIPS mode?	Remarks
Castor XML to Java FC	yes	Provides no cryptography features
EMF XMLToSDO	yes	Provides no cryptography features
EMF SDOToXML	yes	Provides no cryptography features
AssemblyLine FC	yes	Operates as a Server API client
Java Class Function Component	depends	Depends on the FIPS compliance of the Java class, whose method will be invoked by the Function Component. If the Java class does not use cryptography (SSL, encryption, signing, cryptographic hash functions, and so forth) it can be safely used in FIPS mode.
Parser FC	depends	Depends on the FIPS compliance of the Parser that is configured for the Function Component
CBE Generator Function Component	yes	Provides no cryptography features
SendEMail Function Component	yes	Uses default JSSE factories for SSL
Memory Queue FC	depends	Depends on the FIPS compliance of the JDBC driver used by the System Store. (The Memory Queue uses the System Store for persistence.) See “Connectors, Function Components, Parsers” on page 154 for a discussion on the FIPS conformance of JDBC drivers.
Axis Java To Soap FC	yes	Provides no cryptography features
WrapSoap FC	yes	Provides no cryptography features
Invoke Soap WS FC	yes	Uses default JSSE factories for SSL
Axis Soap To Java FC	yes	Provides no cryptography features
Axis EasyInvoke Soap WS FC	yes	Uses default JSSE factories for SSL
Complex Types Generator Function Component	yes	Provides no cryptography features
Remote Command Line Function Component	depends	The cryptographic capabilities of the RXA toolkit are not FIPS compliant. If no cryptography features are used, the component can be used in FIPS mode.
z/OS TSO/E Command Line FC	depends	Depends on the FIPS compliance of the cryptography involved in the TSO command that is invoked by the Function Component
SAP ABAP Application Server Component Suite	no	The SAP cryptographic module has not been FIPS 140-2 certified. If no cryptography features are used, the components can be used in FIPS mode.
Parsers	yes	None of the Tivoli Directory Integrator Parser components use cryptography so all of them can be used in FIPS mode.

Setting com.ibm.di.server.fipsmode.on:

To enable FIPS mode in Tivoli Directory Integrator you must specify it in a property in `global.properties` or `solution.properties`. The property is named `com.ibm.di.server.fipsmode.on` and

can be set to either **true** or **false**. When this property is set to **true**, the TDI Server runs in FIPS mode. In this mode, the IBM FIPS security provider is set in the Tivoli Directory Integrator JVM before the IBM JCE security provider in the providers list. When the TDI FIPS enabling property is true, it also enables FIPS mode in the IBM JSSE2 provider and sets the default JSSE SSL socket factories to be the ones from the IBM JSSE2 provider. By default FIPS mode is not enabled in TDI, that is, the `com.ibm.di.server.fipsmode.on` property is set to **false**.

Using crypto algorithms in FIPS mode:

Only FIPS-compliant crypto algorithms can be used. This means that you must use only FIPS-compliant algorithms in order to stay in FIPS-compliant mode. Using other algorithms violates FIPS compliancy.

Setting `com.ibm.di.securityTransformation`:

When opening an encrypted configuration, TDI uses the `com.ibm.di.securityTransformation` property to get the algorithm that decrypts the configuration. If this property is set to an algorithm which is not FIPS-compliant, and the Tivoli Directory Integrator Server FIPS mode is turned on, then an Exception is thrown. The Exception message which is shown would look something like this:

`CTGDIC012E Could not load file<FILE_PATH>. No such algorithm: <ALGORITHM_NAME>.`

In order to avoid this Exception, always set FIPS-compliant algorithms for this property when running in FIPS mode. By default the `com.ibm.di.securityTransformation` property is set to DES/ECB/Nopadding which is **not** a FIPS-compliant algorithm. This property also defines a cipher for the password-based encryption and decryption of TDI configurations.

Setting properties automatically when running in FIPS mode:

- Tivoli Directory Integrator sets a relevant System property which is not present in the `global.properties` file by default. This property is called `com.ibm.di.cryptoProvider` and is set to the IBMJCEFIPS security provider when run in FIPS mode. You should note that if this property is set in `global.properties` then that particular value is used; if this property is set to a non-FIPS compliant provider, then even if TDI is run in FIPS mode, TDI is **not** FIPS-compliant.
- When in FIPS mode, specific JSSE Socket Factories are used. These are the IBMJSSE2 Socket Factories. This is done automatically by the TDI Server which sets the `ssl.SocketFactory.provider` and the `ssl.ServerSocketFactory.provider` properties to the JSSE implementation classes in IBMJSSE2 provider.

Using the `create stash` command line tool in FIPS mode:

To create a stash file that conforms to FIPS 140-2 standards you must provide the IBMJCEFIPS provider class as the third parameter when using the `createteststash` file tool. For example:

`TDI_install_dir\bin\createteststash Password Password com.ibm.crypto.provider.IBMJCEFIPS`

Using alternatives to RSA encryption in FIPS mode:

In FIPS mode, configure Tivoli Directory Integrator to use the Advanced Encryption Standard (AES) instead of the RSA encryption algorithm. A secret key cipher that is FIPS 140-2 compliant is required. As an acronym, RSA stands for Rivest, Shamir, and Adelman, the inventors of the algorithm. The RSA algorithm is a strong encryption algorithm used for sending data over the internet. The RSA cipher is allowed only to encrypt and decrypt keys for transport (SSL, TLS) to stay within the boundaries of the Approved Mode of FIPS 140-2 Level 1, as stated at: <http://www.ibm.com/developerworks/java/jdk/security/60/FIPShowto.html>.

Running auxiliary tools in FIPS mode: This section provides command line syntax for identifying the appropriate crypto provider, and when generating a secret key.

createstash:

Pass the FIPS 140-2 certified crypto provider IBMJCEFIPS as an explicit provider parameter on the command-line:

```
createstash mypass mykeypass com.ibm.crypto.provider.IBMJCEFIPS
```

cryptoutils:

Pass the FIPS 140-2 certified crypto provider IBMJCEFIPS as an explicit provider on the command-line using the *cryptoproviderclass* option like this:

```
cryptoutils -input registry.txt -output registry.enc -mode encrypt -keystore ../testserver.jks -storepass server  
-alias server -cryptoproviderclass com.ibm.crypto.fips.provider.IBMJCEFIPS
```

Configuring FIPS properties for TDI: Running keytool/Ikeyman in FIPS mode:

To use the keytool and Ikeyman utilities in FIPS mode, edit the java.security file in *TDI_install_dir/jvm/jre/lib/security*. In the first two lines in the java.security file, set the IBMJCEFIPS provider first and the IBMJCE security provider second. For example:

```
security.provider.1=com.ibm.crypto.fips.provider.IBMJCEFIPS  
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

However, on Solaris and HP-UX, the SUN provider should always be the first provider in the security providers list.

Configuring SSL and PKI certificates

Tivoli Directory Integrator uses both Secure Socket Layer (SSL) and Public Key Infrastructure (PKI) encryption methods. SSL and PKI provides an important foundation for many of the TDI and TDI server features. SSL provides for encryption and authentication of network traffic between two remote communicating parties. Similarly, PKI (public key infrastructure) enables users of unsecured networks to securely and privately exchange data by using a public and a private cryptographic key pair that is obtained and shared through a trusted authority. See “Working with certificates” on page 162.

SSL certificate:

An SSL certificate resides on a secure server and is used to encrypt the data that identifies the server. The SSL certificate helps to prove the site belongs to the entity who claims it and contains information about the certificate holder, the domain that the certificate was issued to, the name of the Certificate Authority who issued the certificate, and the root and the country it was issued in.

PKI certificate:

A PKI certificate enables users of an unsecured network to add security and privacy to data exchanges. PKI uses a cryptographic key pair that it gets and shares through a trusted authority called a Certificate Authority (CA). Using PKI, you can obtain a certificate that can identify an individual or an organization and directory services that can store the certificates. The CA can also revoke the certificates when necessary. The most common use of a digital certificate is to verify that a user sending a message is who the sender claims to be, and to provide the receiver with the encryption of the reply.

Follow these steps to provide separate configuration options for certificates to be used for PKI Encryption and SSL:

1. Add the following properties:

```
com.ibm.di.server.encryption.keystore  
com.ibm.di.server.encryption.key.alias  
api.keystore.password  
api.key.password
```

2. Rename the following properties as shown:

```
com.ibm.di.server.keystore ----- > api.keystore  
com.ibm.di.server.key.alias ----->api.key.alias
```

Note: The `theidisrv.sth` file now holds the password only for the encryption file.

Encrypting and decrypting using CryptoUtils

Using Tivoli Directory Integrator, you can PKI encrypt sensitive properties in the `global.properties` or in the `solution.properties` file. One method of decrypting PKI-encrypted properties is to use the Configuration Editor (CE) properties editor. The CryptoUtils command-line utility is another method of decrypting PKI-encrypted properties files. Decryption requires you to give your PKI credentials so that unauthorized users cannot access sensitive information. You can properties files that contain PKI encrypted properties using CryptoUtils. see “The TDI Encryption utility” on page 126.

Working with certificates

Someone who wishes to send an encrypted message applies for a digital certificate from a CA. The CA issues an encrypted digital certificate that contains the applicant's public key and other identification data. The CA makes reveals its own public key through printed media or perhaps on the Internet. The recipient of an encrypted message uses the CA's public key to decode the digital certificate attached to the message, verifies the certificate as issued by the CA, and then gets the sender's public key and identification data from the certificate. With this information, the recipient can send an encrypted reply.

Digital certificates are of two types:

- CA-signed certificates
- Self-signed certificates

CA-signed certificates are signed by a Certificate Authority such as VeriSign and thawte. A self-signed certificate is an identity certificate that is signed by its own creator.

Comparing CA-signed and Self-signed certificates

Certificate authority signed certificates:

Certificate authorities such as VeriSign require a procedure whereby applicants can prove their identities and obtain certificates that authenticate both the identity of the certificate applicants and its own identity as a signer of a certificate.

Self-signed certificates:

Typically there is a local certification authority (CA), that is, the certificates do not come from any of the well known CAs like VeriSign, and so on. 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 truststore of each server or client that is using SSL.

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 truststores. If a client **C** connects to a server **S**, **C** must have **S**'s self-signed certificate in its truststore. If a client **C** does PKI authentication (symmetric SSL) to a server **S**, **S** must have **C**'s self-signed certificate in its truststore. Note: Self-signed certificates can be used for either a client or a server certificate. See the “Manage keys, certificates and keystores” on page 89 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.

Configuring certificates using PKI and SSL

Tivoli Directory Integrator provides separate configuration options for certificates to be used for public key infrastructure (PKI) encryption and Secure Socket Layer (SSL) connection. Independent configuration of PKI and SSL certificates allows you to migrate your encrypted properties separately from the process of upgrading your SSL certificates.

Under PKI, a Certificate Authority (CA) binds public keys to user identities. The user identity must be unique for each CA. Public Key certificates collect each user, user identity, public key, their binding, validity conditions, and other attributes that are made unforgettable in public key certificates issued by the CA.

The certificates used for SSL may expire, or for security reasons, SSL certificates may have to be refreshed frequently. Certificates used for PKI encryption can be persisted longer than it is appropriate to persist SSL certificates. PKI certificates should be maintained in case there is data that has been encrypted using the public key certificate. As a result, Tivoli Directory Integrator allows you to configure PKI and SSL certificates separately. Each server for an SSL connection and each client performing PKI authentication must issue a request for a certificate to the local CA, and must add the resulting certificate into its keystore.

These properties are added to the `global.properties` file:

```
com.ibm.di.server.encryption.keystore  
com.ibm.di.server.encryption.key.alias
```

These properties variables are set to the same values as the ones already in `global.properties`:

```
api.keystore=truststore  
api.key.alias=server
```

Using cryptographic keys located on hardware devices

The RSA signing and encryption algorithm (developed by Ron Rivest, Adi Shamir, and Leonard Adleman) is a well-known public key cipher. RSA Laboratories (Part of EMC Corp.) have published the PKCS#11 standard, which defines a platform-independent API to hardware cryptographic tokens, such as Hardware Security Modules and smart cards. The PKCS#11 API defines most commonly used cryptographic object types, including:

- RSA keys
- X.509 Certificates
- Data Encryption Standard (DES)DES/Triple DES keys
- All the functions required for using, creating or generating, modifying, and deleting the above objects

Public-Key Cryptography Standards (PKCS) PKCS#11 is a standard that provides a common application interface to cryptographic services on various platforms using various hardware cryptographic devices. Hardware Cryptographic key storage devices allow keys to be stored on hardware devices. IBM Tivoli Directory Integrator supports private keys and certificates on crypto devices that are PKCS#11 compliant. Support is provided on all hardware devices supported by the IBM Java PKCS libraries shipped with the IBM Java Runtime Environment (JRE). PKCS standards are a set of common protocols that allow secure information exchange over networks using a public key infrastructure (PKI). IBM Tivoli Directory Integrator can store Secure Socket Layer (SSL) keys on the hardware devices. For the requirement to store keys on hardware devices, the following new properties are available in the `global.properties` file:

```
##PKCS11 options  
##Set the value of following properties to use PKCS11 enabled devices to store TDI servers private key /  
##certificate.  
com.ibm.di.pkcs11cfg=etc\pkcs11.cfg  
com.ibm.di.server.pkcs11=false  
com.ibm.di.server.pkcs11.library=  
com.ibm.di.server.pkcs11.slot=  
{protect}-com.ibm.di.server.pkcs11.password=PASSWORD
```

The default value of the property `com.ibm.di.server.pkcs11` is false. The value corresponding to the property `com.ibm.di.server.pkcs11.password` is encrypted.

Using IBMPCKS11 to access devices and to store SSL keys and certificates

IBM Tivoli Directory Integrator uses IBMPCKS11 to access crypto hardware devices that store the SSL keys and certificates. Support is provided for all hardware devices supported by the IBM Java PKCS libraries and shipped with the IBM JRE.

Table 21. SSL supported properties

Property	Default value	Description
<code>com.ibm.di.pkcs11.cfg</code>	<code>etc\pkcs11.cfg</code>	Use CFG file to point to the path of the configuration file required to initialize the IBM PKCS11 implementation provider.
<code>com.ibm.di.server.pkcs11</code>	false	Use PKCS#11 compliant crypto devices for ssl.
<code>com.ibm.di.server.pkcs11.library</code>		Use this property to specify the path to the PKCS11client library.
<code>com.ibm.di.server.pkcs11.slot</code>		Specify the slot number of the device.
{protect}- <code>com.ibm.di.server.pkcs11.pass</code>		Use this password to access the pkcs11 compliant crypto device.
<code>com.ibm.di.server.pkcs11.accl</code>	false	Use =true to set hardware cryptographic devices for cryptographic operations.

Enabling or disabling padding

Padding means adding extra bits to a transmission so that the transmission is the exact, required, size. Some encryption and decryption algorithms require their input to be an exact multiple of the block size. If the plaintext to be encrypted is not an exact multiple, you must *pad* before encrypting by adding a padding string. When decrypting let the receiving party know how to remove the padding.

Note: All properties listed in the `global.properties` file can be set in the configuration file by the same name; it is recommended that you edit your `solution.properties` file instead if you have one.

These properties can be protected by encryption using the {protect}- prefix (see section “Standard TDI encryption of `global.properties` or `solution.properties`” on page 125 for details).

When setting the property for padding, the default value is DES/ECB/NoPadding. The padding property defines an algorithm or cipher for password-based encryption and decryption of Tivoli Directory Integrator configurations. The property is: `com.ibm.di.securityTransformation`.

Maintaining encryption artifacts – keys, certificates, keystores, encrypted files

Note: The default SSL certificates of IBM Tivoli Directory Integrator are changed in v7.1.1. Therefore, IBM Tivoli Directory Integrator v7.1.1 fails to decrypt the following items, if encrypted using the default TDI certificates:

- encrypted passwords in global or solution.properties
- protected properties in external property stores
- encrypted TDI configuration XML files in previous versions

Therefore, decrypt all the encrypted passwords using your previous version of IBM Tivoli Directory Integrator. For more information about encryption utility, see “The TDI Encryption

utility" on page 126. Once the passwords are retrieved as text, use them in the latest version. The passwords are encrypted with the new default certificates once the server or Configuration Editor is started.

Changed encryption key

Any change of the key that the Server uses for encryption leads to a need for migration of existing encrypted files. To migrate an encrypted file, you should decrypt it with the old encryption key and encrypt it with the new one. Encryption and decryption can be done using the cryptoutils tool.

Files which are often encrypted or contain encrypted parts are: configurations, the User Registry and properties files (Tivoli Directory Integrator properties files can contain encrypted properties, although the files are usually not encrypted as a whole).

Note: By default all sensitive properties (such as passwords) inside `global.properties` or `solution.properties` are encrypted. As a rule of thumb you should always migrate `global.properties` and `solution.properties` files when you change the Server encryption key.

Changed password for encryption key or keystore

The Server reads the password for the keystore that holds the encryption key and the password for the encryption key itself from the Server stash file. Thus if any of those passwords is changed, the stash file must be updated. This can be done using the `createtestash` tool.

Expired encryption certificate

If the Server uses public-key encryption, the certificate associated with the encryption key-pair can potentially expire at some point in time. If this happens, the certificate can be renewed using the procedure described in section "Extend the validity of a certificate using keytool". That procedure preserves the underlying keys, so no migration of existing encrypted files is necessary.

Chapter 10. Configuring the TDI Server API

The IBM Tivoli Directory Integrator 7.1.1 Server API provides a set of programming calls that can be used to develop Tivoli Directory Integrator solutions and interact with the server locally and remotely. It also includes a management layer that exposes the Server API calls through the Java Management Extensions (JMX) interface. This section provides information about properties you can use to configure the Server API.

- For information on using the Server API, see "Appendix C. Server API" in the *IBM Tivoli Directory Integrator V7.1.1 Reference Guide*.
- For additional information on configuring the server API, see "Configuring the Server API" on page 101.
- For information on Tivoli Directory Integrator server security, see "Remote Server API" on page 100.

Server ID

Remote clients can identify the server they are talking to if the server can be distinguished using a unique ID. Tivoli Directory Integrator allows you to specify unique server IDs that allow remote clients, such as the Administration and Monitoring Console (AMC), to connect to different Tivoli Directory Integrator servers at different times using the same IP and port. In order to connect using the same ID and port but at different times, Tivoli Directory Integrator client applications must be able to register these client applications as different Tivoli Directory Integrator servers that you can associate with different data and databases.

Users are to assign the unique IDs manually, ensuring that any remote client (such as AMC) can connect to a Tivoli Directory Integrator server based on the IP address, the port, and the unique ID of the Tivoli Directory Integrator server. AMC registers every server with a unique ID, so that a Tivoli Directory Integrator server cannot be registered more than once by mistake or intentionally. When assigning IDs manually, users must ensure that different Tivoli Directory Integrator servers have distinguishable IDs.

You can configure the unique server ID property using `com.ibm.di.server.id`. To give a server a unique server id for a given server, provide a unique ID string for this property in the `global.properties` or `solution.properties` file on the server you are identifying. The default value for `com.ibm.di.server.id` is blank.

Exception for password protected Configs

The server API throws an exception if you use a password protected Config without using a password. The Tivoli Directory Integrator server API can detect and handle server problems when the server is faced with clients trying to access password protected configurations without supplying a password. A message displays, notifying the user about the problem. The error message is invoked when no password is supplied or the when the password entered is wrong. See "AMC and encrypted configs" on page 228.

Server RMI

Due to increasing needs for remote access to each Tivoli Directory Integrator server, Remote Method Invocation (RMI) is enabled by default. To ensure adequate security, default remote access requires Secure Socket Layer (SSL) for client authentication. The SSL access is facilitated with the sample keystore and truststore that are deployed with Tivoli Directory Integrator. See "SSL client authentication" on page 96 and "Summary of Server API Authentication options" on page 112.

Config load time-out interval

If a config instance does not completely load the configuration file when the server API makes a call, the server API returns a null object. You can add a time-out interval by adding the following property: `api.config.timeout` in the `global.properties` file. The interval for loading the configuration file is set to two minutes by default. If the config file does not load within the time interval, an exception is thrown.

Chapter 11. Properties

You can use Properties to configure TDI components and the TDI Server. Properties are simple keyword:value pairs of parameters kept outside your configuration files (configs), stored in External Properties files. This enables you to keep confidential information like passwords outside of your Config files. The `global.properties` file is the main configuration file for TDI. Properties are defined in the `global.properties` file or in the `solution.properties` file. The `solutions.property` file is a writable copy of the `global.properties` file and is used when the server is started from the solution directory. If a solution directory different from the installation directory is specified during installation, then a copy of the `global.properties` file named `solution.properties` is created in the Tivoli Directory Integrator solution directory. Both files are text files, and are written so that they can be understood by the operating system that is running on the platform.

Properties are single-valued data containers that hold parameter information, for example, `true` or `5000`. You can access properties from script using entry functions like `getProperty()` and `setProperty()`. Get and set methods work directly with property values. Entry objects can also contain properties. Like Attributes, properties are data containers. Attributes are used to store data content, but properties hold parametric information. Property values and Attributes can be any type of Java Object. Properties do not show up in:

- Attribute map selection.
- Work entry lists.

Working with properties

This section introduces the primary concepts you need when working with properties. Properties set in any properties files form a baseline for the entire IBM Tivoli Directory Integrator installation for all users on that computer. 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 override anything set in any of the global installation property files, including the `global.properties` and `solution.properties` files. Furthermore, 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:

- Set the environment variable `TDI_SOLDIR` before starting the Configuration Editor or the Server.
- Specify the `-s` parameter to the `ibmditk` script to start the Configuration Editor or the `ibmdisrv` script to start the TDI Server. This takes precedence over setting `TDI_SOLDIR`. If `TDI_SOLDIR` equals the installation directory, all property files is 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 makes a copy of all the property files into your Solutions Directory (it does 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` is 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 are copied under their own names.

For documentation purposes, the original `global.properties` file from the installation directory is copied to the `<Solution directory>/etc` folder; this file is **not** used for any other purpose.

Migrating using properties and the tdimiggbl tool

The tdimiggbl tool helps you to migrate your global.properties file from one version of Tivoli Directory Integrator to a higher version. See the following chapter Chapter 5, "Migrating," on page 63.

Global properties

Global properties are used to configure the Tivoli Directory Integrator Server settings that are kept in a file called global.properties in the etc folder of your installation directory. All properties included in the global.properties file are listed with their default values and explained in this chapter. A reference to more detailed documentation is provided, where possible, in the beginning of the groups of properties. The Configuration Editor (CE) (ibmditk) and the Tivoli Directory Integrator server (ibmdisrv) read the global.properties file on startup. This file is read and applied before a file called solution.properties from your Solution Directory is read.

Table 22. Some important Tivoli Directory Integrator global properties

Use of the property	Property	Default value	Description
Add your own .jar or .zip files	com.ibm.di.loader.userjars	c:\myjars	Specifies directories or jar files, separated by the Java Property "path.separator", which is ":" on Linux and ";" on Windows. Directories are searched recursively by the TDILoader for jar files containing classes and resources. Only files with a .zip or .jar extension are searched.
Define cipher	com.ibm.di.securityTransformation	DES/ECB/NoPadding	Defines a cipher for the password-based encryption or decryption of Tivoli Directory Integrator configurations. Changed in Tivoli Directory Integrator 7.0.
Enable config autoload	com.ibm.di.server.autoload	autoload.tdi	Looks for *.xml files in the directory specified by the "ibmdisrv -d" command. Executes each *.xml file found in the directory defined by -d.

Solution properties

Solution properties typically override global properties and are found in a file in your solution directory called solution.properties. The solution.properties file is by default a copy of the global.properties file, and you should edit the solution.properties file when configuring TDI, because it is read last out of all the properties files. If you want to, you can delete properties in your solution.properties file and add property configuration statements that you specifically want to override the global.properties defaults.

Java properties

Java properties are variables and settings of the Java Virtual Machine (JVM). Java log (Jlog) file properties are shown in "Useful JLOG parameters" on page 211.

Note: 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.

Table 23. Java properties

Property	Default value	Description
javax.net.debug	none	Sets debug mode for the JSSE provider.
com.ibm.di.javacmd	none	Overrides the Java interpreter.
com.ibm.di.installdir	none	Uses this path to the Java executable file when running AssemblyLines from the Configuration Editor.
com.ibm.di.jvmdir	TDI_root/jvm	Defines the directory path where the JRE that Tivoli Directory Integrator uses is installed.
com.ibm.di.server.maxThreadsRunning	500	Sets this number of threads Tivoli Directory Integrator. Must be set higher than 3 to have any effect.
com.ibm.di.server.securemode	false	Sets the mode in which Tivoli Directory Integrator is running. (standard or secure)
com.ibm.di.server.keystore	testserver.jks	Names the keystore of the Server's SSL certificate. Renamed in Tivoli Directory Integrator 7.0.
com.ibm.di.server.key.alias	server	Names the key alias of the Server's SSL certificate. Renamed in Tivoli Directory Integrator 7.0.
{protect}-api.keystore.password	server (encrypted by default)	Provides the password for the server API keystore. Added in TDI 7.0.
{protect}-api.key.password		Provides the key password. If not specified, uses server keystore password. Added in Tivoli Directory Integrator 7.0.
com.ibm.di.server.encryption.keystore	testserver.jks	Names the keystore of the server encryption key. Added in Tivoli Directory Integrator 7.0.
com.ibm.di.server.encryption.key.alias	server	Provides the key alias of the server encryption key. Added in Tivoli Directory Integrator 7.0.
com.ibm.di.server.encryption.keystoretype	jks	Provides the type of the keystore that hosts the key used by the server for encryption. Added in Tivoli Directory Integrator 7.0.
com.ibm.di.server.encryption.transformation	RSA	Names the cryptographic transformation used by the server for encryption. Can be set to either "RSA" (public key encryption) or to some secret key transformation. Added in Tivoli Directory Integrator 7.0.

Table 23. Java properties (continued)

Property	Default value	Description
com.ibm.di.server.fipsmode.on	false	Enables or disables FIPS standards in TDI. If this property is set to true, TDI runs in FIPS-compliant mode. For more information on FIPS mode, see Added in Tivoli Directory Integrator 7.0.
com.ibm.di.default.bind.address	*	The default bind address for the whole TDI Server - the components and the Server API.

System properties

System properties are stored in the System Store instead of being stored in an external properties file such as `solution.properties`. Certain system properties and Java properties are read-only. These system properties are shown in the respective Property Stores (for example, System Store). Attempting to modify these read-only properties has no effect. See also Chapter 12, “System Store,” on page 173.

Chapter 12. 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 a relational database implemented fully in Java, known as Apache Derby, and previously known as Cloudscape.

The System Store supports the following objects:

- Delta Tables
- User Property Store
- Password Store

Default location of System Store:

The default location of IBM Tivoli Directory Integrator System Store database, in network mode, is your Solution Directory. Therefore, you can have a System Store for each of your Solution Directories.

To share a single System Store across all the Solution Directories, replace \$soldir\$ value with the actual *TDI_install_dir* in the com.ibm.di.store.database property of global.properties file and solution.properties file, if already created.

When you create a Solution Directory, update the following properties in the solution.properties file with unique values to avoid conflicts with other Solution Directory settings:

- com.ibm.di.store.port=1527
- api.remote.naming.port=1099
- web.server.port=1098
- System Queue port or Active MQ port in <soldir>/etc/activemq.xml

Note:

The following example describes the effect of specifying the same value for com.ibm.di.store.port property across multiple Solution Directories.

There are two solution directories (soldir1, soldir2) with the same com.ibm.di.store.port values (1527, 1527), and with unique api.remote.naming.port values (1099, 41099).

When you start server on soldir1, the server starts on port 1099 and System Store on port 1527, inside soldir1.

When you start Server on soldir2, the server starts on port 41099 and connects to the System Store, which is already listening on port 1527 inside soldir1.

Property stores

Password store and User property stores are types of system stores.

Password Store

The *Password Store* is an external repository that stores a value which results from changing the value for a password syntax component. The password protection mechanism is directly related to the

configuration windows offered to the user. The configuration windows, or forms, contain descriptions of each parameter and its syntax. One type of syntax is *password* which causes the Configuration Editor to use a password text field for editing. 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 is 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 is 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.

User property stores

The User Property Store is a System Store table used for maintaining serialized Java objects associated with a key value. This is where persistent component parameters and properties (such as the **Iterator State Store**) are maintained, as well as any data you store. The System Store implements User property stores as one of its three types of persistent stores for IBM Tivoli Directory Integrator components. For information on TDI user interfaces that allow you to select properties from a property store, see “Add a Solution View” on page 237.

Third-party RDBMS as System Store

The System Store can also be configured to use other multi-user RDBMS systems, as opposed to using the bundled database, Apache Derby. 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_dir/etc` directory, for the supported configurations of IBM DB2, Oracle and MS SQL*Server.

It is also possible to take advantage of suitable templates built in to the Configuration Editor, by going to the appropriate Tivoli Directory Integrator Server document. Right-click on the Server in the Servers pane, and select **Edit system store settings**. The **Server System Store** header in the window is a context-sensitive menu; it has selections for Derby Embedded, Derby Networked, Oracle, DB2, MS SQL*Server 2005+ and IBM SolidDB.

Note: A System Store can also be configured on a per-project basis in the Configuration Editor; these settings are then stored in the Config file when the project is exported, and take precedence over the System Store defined for the Server.

JDBC Driver parameters provide a path to the database; additional properties are used to specify tailored SQL for certain operations Tivoli Directory Integrator 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} => is 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 section lists 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_install_dir/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.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

Use of MS SQL Server requires that you install a number of Microsoft client libraries in the *TDI_install_dir/jars* directory.

JDBC connection parameters

```
com.ibm.di.store.database=jdbc:Microsoft:sq1server://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 2008, the driver jar file to be placed in the *TDI_install_dir/jars* directory is sqljdbc.jar (only one file is required) and it can be obtained from your SQL Server 2008 installation at <Microsoft SQL Server 2005-Install-Dir>/sqljdbc_<version>/<language>/sqljdbc.jar; the JDBC connection parameters need to be specified as follows:

```
com.ibm.di.store.database=jdbc:sq1server://localhost:1433;DatabaseName=name;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 selectMethod property is optional to the jdbc URL. When this property is set to "cursor", a database cursor is created. This is useful when reading very large result sets that cannot be contained in the clients memory.

The default behavior of selectMethod is not "cursor", but "direct", which keeps result sets in clients memory, thus providing much faster performance. So unless memory is a problem, it is better to use the default "direct" behavior. For more information:
[http://msdn.microsoft.com/en-us/library/ms378988\(SQL.90\).aspx](http://msdn.microsoft.com/en-us/library/ms378988(SQL.90).aspx).

JDBC connection parameters (for JSQlConnect driver)

```
com.ibm.di.store.database= jdbc:JSQlConnect://itdiderver/database=reqpro
com.ibm.di.store.jdbc.driver= com.jnetdirect.sql.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_install_dir/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.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/iddidb
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 *iddidb* 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.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.sandbox.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL,
ENTRY BLOB )

```

IBM SolidDB

IBM SolidDB requires that the SolidDriver2.0.jar file is put in the *TDI_install_dir/jars* directory. This JAR can be obtained from the SolidDB installation (from *SolidDB_install_dir/jdbc/SolidDriver2.0.jar*.

JDBC connection parameters

```

com.ibm.di.store.database=jdbc:solid://localhost:1315
com.ibm.di.store.jdbc.driver=solid.jdbc.SolidDriver
com.ibm.di.store.jdbc.urlprefix=jdbc:solid:
com.ibm.di.store.jdbc.user=dba
{protect}-com.ibm.di.store.jdbc.password=dba

```

Create table statements

```

com.ibm.di.store.create.delta.systable=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH)
PRIMARY KEY NOT NULL, SEQUENCEID int, VERSION int)
com.ibm.di.store.create.delta.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH)
PRIMARY KEY NOT NULL, SEQUENCEID int, ENTRY BLOB )
com.ibm.di.store.create.property.store=CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH)
PRIMARY KEY NOT NULL, ENTRY BLOB )
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)

```

Using Derby to hold your System Store

The remainder of this chapter discusses the operational aspects of using Derby, in particular in conjunction with using Derby to hold your System Store.

Note: With regards to third party RDBMSs, in order to hold encrypted password values you may have to size 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.

Derby can run in either of two modes: *embedded* and *networked*. By default, as specified in the `global.properties` file, Derby runs in *networked* mode.

The System Store used by Tivoli Directory Integrator releases before V7.0 was Derby (then called Cloudscape) in *embedded* mode. There are drawbacks to the way Derby runs in embedded mode. In embedded mode, Derby runs as a separate thread within the JVM when required. Startup and shutdown of Derby is automatic in embedded mode. However, when run this way, this Derby thread claims exclusive access to the database files. This can become problematic when different JVMs, each with its own Derby thread, try to access the same System Store.

In embedded mode, these actions cause 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.
- Startup of the Configuration Editor (GUI)
- Startup of an AssemblyLine from within the Configuration Editor

None of these actions by themselves causes the Derby thread to start. However, the Derby 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 Delta Tables and the User Property Store).

The solution to the access conflicts as outlined previously is to run Derby in *networked* mode, which enables concurrent access to the System Store. Also enable user authentication in derby to avoid security concerns in networked mode. To provide security at the database level, TDI uses the BUILTIN security provider for Derby. BUILTIN ensures that only valid users are able to access the Derby database. When you have Derby configured in networked mode, you can work with multiple instances of Derby databases booted as System Stores. You can also configure a Derby instance to work with a specific Configuration file instance.

Note: Depending on how Derby was started, instances of Derby can be left running in networked mode, even after all other Tivoli Directory Integrator processes have terminated.

When you set the property `derby.drda.startNetworkServer` to true (by default, this is the case, in `global.properties`), the Network Server automatically starts when you start Derby (in this context, Derby starts when the embedded driver is loaded). You may have to terminate Derby manually, if desired.

Configuring Derby Instances

To configure and manage multiple Derby instances and to provide facilities to start, stop and restart Derby servers in networked mode a menu option called **System Store** is provided in the Tivoli Directory Integrator Configuration Editor, as part of **Solution Logging and Settings** configuration of a project. Many of the configuration options listed take default values from the `global.properties` file, which was the configuration base for previous versions of Tivoli Directory Integrator; now.

The **System Store** menu option also provides ways to configure the System Store to use other databases like IBM DB2 as the backend RDBMS. For more information, refer to "System Store settings" under **The Configuration Editor -> Solution Logging and Settings** in *IBM Tivoli Directory Integrator V7.1.1 Users Guide*.

Starting Derby in networked mode

If the `com.ibm.di.store.hostname` property is set to `localhost` then remote connections are not allowed. If the `com.ibm.di.store.hostname` property is set to the IP address of the local computer running Tivoli Directory Integrator, then remote clients can access this Derby instance by using the IP address. You can only start the network server for the local computer.

Table 24. Starting Derby in networked mode

Property	Default value	Description
<code>com.ibm.di.store.start.mode</code>	automatic	The mode for starting up the Derby server process when required – set to automatic or manual.
<code>Com.ibm.di.store.hostname</code>	<code>localhost</code>	The URL of the Derby server.
<code>Com.ibm.di.store.port</code>	1527	The port for connecting to the Derby server.
<code>Com.ibm.di.store.sysibm</code>	true	The state for using the SYSIBM schema or not; values true or false.
<code>com.ibm.di.store.varchar.length</code>	512	The varchar(<code>length</code>) for the ID columns used in system store and System Store (PES) connector tables.
<code>com.ibm.di.store.database</code>	<code>jdbc:derby://localhost:1527/\$soldir\$/TDISysStore;create=true</code>	Sets your Solution Directory as default location of the System Store database. Note: Do not replace <code>\$soldir\$</code> value with <code>absolute</code> path of Solution Directory. The path is automatically updated at runtime in JVM

Enabling user authentication in System Store

Add these properties to the `global.properties` file after the System Store network mode properties.

Table 25. Enable user authentication in System Store

Property	Default value	Description
<code>derby.connection.requireAuthentication</code>	true	Enables user authentication for the System Store.
<code>derby.authentication.provider</code>	BUILTIN	Sets the user authentication provider to BUILTIN. This is the most basic and simple authentication provider that Derby has.
<code>derby.database.defaultConnectionMode</code>	fullAccess	Defines the access level to the System Store user. The different access levels supported by Derby are "fullAccess", "readOnly" and "noAccess".

Create statements for System Store tables

You can configure create table SQL statements for

- Delta systable
- Delta table
- Property table
- Sandbox tables
- Record AssemblyLine table
- Tombstone manager table
- `ibmsnap_commitseq` column name

Table 26. Create statements for System Store

Property	Default value	Description
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)	Create table SQL statements for the delta systable.
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)	Create table SQL statements for the delta table.
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)	Create table SQL statements for the property table.
com.ibm.di.store.create.sandbox.store	CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ENTRY BLOB)	Create table SQL statements for the Sandbox tables.
com.ibm.di.store.create.recal.conops	CREATE TABLE {0} (METHOD varchar(VARCHAR_LENGTH), RESULT BLOB, ERROR BLOB)	Create table SQL statements for Record AL.
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_TIME TIMESTAMP, CREATED_ON TIMESTAMP, COMPONENT_NAME VARCHAR(1024), CONFIGURATION VARCHAR(1024), EXIT_CODE INT, ERROR_DESCR VARCHAR(1024), STATS LONG VARCHAR FOR BIT DATA, GUID VARCHAR(1024) NOT NULL, USER_MESSAGE VARCHAR(1024), UNIQUE (ID, GUID))	Specify the SQL statement for creating the Tombstone Manager table. Keep the same table names and field names.
com.ibm.di.conn.rdbmschlog.cdcolname	ibmsnap_commitseq	Provide the ibmsnap_commitseq column name to be used by the RDBMS changelog connector.

Backing up Derby databases

Another matter that needs to be given some thought is **backup** of the data contained in a Derby database. The recommended (and simplest) way of doing this is to

- Shutdown the Derby database (if running in embedded mode, shut down all Tivoli Directory Integrator instances and Configuration Editor instances)

- Copy the entire Derby directory in your Tivoli Directory Integrator home directory (or whatever Derby directory your `global.properties` file is pointing to) to a different location, and ensure that this data is safe
- Restart the Derby 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 Derby issues

This section does not attempt to be a comprehensive Troubleshooting Guide for Derby, but there are a number of symptoms that are observed sometimes in the context of usage of Derby as the underlying database in Tivoli Directory Integrator. These are:

Schema 'SYSIBM' does not exist error

Question:

I'm trying to use Derby 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. Note that for a networked mode Derby server to open an embedded mode database, the SYSIBM schema MUST be loaded. The SYSIBM schema is a special schema loaded by the Derby server. The SYSIBM contains stored prepared statements that return result sets to determine metadata information.

Corrective action:

To solve this problem start the Derby networked server with the `"-ld"` flag, like:

```
./dbserver start -p 1527 -ld
```

Another Instance of Derby may already be booted

You may get the following error sometimes, especially when using Derby in embedded mode:

```
[ERROR XSDB6: Another instance of Derby may have already booted the database D:\tdi60\Derby.]
```

Explanation:

Derby tries to prevent two instances of Derby from booting the same database (in this case `D:\tdi60\Derby`). This can happen if you are running two AssemblyLines which are trying to update the same Derby 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 Derby database, then the correct mode of Derby should be networked mode; this mode of operation does not have that limitation.
2. You can work around this by closing the database using the **Browse Server Stores** option and then clicking on the **Close** button. Even if the database is not open, just opening and closing again through the **Browse Server Stores** option help solve this problem.

Future versions of Tivoli Directory Integrator attempt to handle this situation automatically, and stop and start Derby as required.

Can I use DB2 as a system store?

In Tivoli Directory Integrator it is possible to use DB2 as a system store, instead of the bundled Derby database system. However, some modification of system properties files is required for this to function correctly. You must replace the section on Derby 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-Derby databases as system store. 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.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 Derby network server?

This may be because the Derby Server has been started by passing "localhost" as the hostname. This disallows any remote connections to be made to Derby. Stop the Derby server and start it with hostname parameter specified as the computer's IP address. This can be done by going to the Configuration Editor's **Server System Store** Server Settings window (available from the context menu on a server in the Servers view).

For more details, check <http://db.apache.org/derby/docs/10.5/adminguide/tadmincbdjhfd.html>

Pre-6.0 (properties file) configuration of Cloudscape

Previous versions of Tivoli Directory Integrator configured the System Store using Cloudscape by means of a set of properties in the `global.properties` file, and version 7.0, now using Derby, 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 Derby Instances" on page 178.

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 (commented out by default) deals with running Derby in embedded (dedicated), non-shared mode.

- The second section (enabled by default) deals with Derby in networked or shared mode. If you determine that you must use dedicated mode, uncomment the first section and comment the second.

Note that the section with configuration parameters for embedded mode specifies a relative path for the Derby database; this typically means the database is 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 must 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 installation directory.

When deploying networked or shared mode, startup of the Derby database thread is automatic; however, shutdown is not. You should shut down the instance when you are finished with your last AssemblyLine.

Cloudscape command-line utility:

To make working with the Derby 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 have to join the middle 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

Stop DBServer ./dbserver shutdown

The full list of sub-commands that you can specify to the dbserver script, and which are sent to Derby is:

- start [-h <host>] [-p <portnumber>]: This starts the network server on the port/host specified or on localhost, port 1527 if no host/port is specified and no properties are set to override the defaults. By default Network Server will only listen for connections from the machine on which it is running. Use -h 0.0.0.0 to listen on all interfaces or -h <hostname> to listen on a specific interface on a multiple IP machine.
- shutdown [-h <host>] [-p <portnumber>]: This shutdowns the network server on the host and port specified or on the local host and port 1527 (default) if no host or port is specified.
- ping [-h <host>] [-p <portnumber>]: This will test whether the Network Server is up.
- sysinfo [-h <host>] [-p <portnumber>]: This prints classpath and version information about the Network Server, the JVM and the Cloudscape server.
- runtimeinfo [-h <host>] [-p <portnumber>]: This prints extensive debugging information about sessions, threads, prepared statements, and memory usage for the running Network Server.

- `logconnections {on | off} [-h <host>] [-p <portnumber>]`: This turns logging of connections and disconnections on and off. Connections and disconnections are logged to `derby.log`. Default is off.
- `maxthreads <max> [-h <host>] [-p <portnumber>]`: This sets the maximum number of threads that can be used for connections. Default 0 (unlimited).
- `timeslice <milliseconds> [-h <host>] [-p <portnumber>]`: This sets the time each session can have using a connection thread before yielding to a waiting session. Default is 0 (no yield).
- `trace {on | off} [-s <session id>] [-h <host>] [-p <portnumber>]`: This turns drda tracing on or off for the specified session or if no session is specified for all sessions. Default is off .
- `tracedirectory <tracedirectory> [-h <host>] [-p <portnumber>]`: This changes where new trace files will be placed. For sessions with tracing already turned on, trace files remain in the previous location. Default is `cloudscape.system.home` .

When running in networked mode, the Derby 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 have 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 Derby 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 Derby home at <http://db.apache.org/derby>, documentation at <http://db.apache.org/derby/manuals/index.html>.

For further information on the command options and the specification on each command, see <http://db.apache.org/derby/docs/10.0/publishedapi/org/apache/derby/drda/NetworkServerControl.html>.

Chapter 13. Command-line options

Command-line options must have their value followed immediately after the option. Do not use a space between the option and the value. There are options for:

- “Configuration Editor”
- “Server” on page 186
- “Command Line Interface – tdisrvctl utility” on page 189

Configuration Editor

The CE is launched using the `ibmditk` wrapper script. This script invokes the Eclipse launcher for Tivoli Directory Integrator (`ce/eclipsese/miadmin`) with the proper settings for the Java VM and the Tivoli Directory Integrator install location property, both of which are required to run the current CE.

The Eclipse launcher (`ce/eclipsese/miadmin`) is a standard Eclipse launcher that takes command line parameters of its own. See [Eclipse Command Line Options](#) for a complete description of the Eclipse command line options.

```
%TDI_HOME_DIR%\ce\.eclipsese\miadmin" -vm "%TDI_JAVA_BIN_DIR%\javaw" -vmargs -Dcom.ibm.di.loader.IDILoader.path="%TDI_HOME_DIR%" %*
```

The above is a fragment of the `ibmditk` script showing the two required parameters (eclipse command line parameters) that the CE needs.

Of notable interest is the “-data” command line option that specifies the location of the workspace to use. If you are going to run multiple instances of the CE, you have to specify a different workspace for each instance of the CE since the workspace is locked by each instance. For example:

```
ibmditk -data c:/instance1_workspace
```

The above command launches the CE using `c:/instance1_workspace` as its workspace location.

Shutdown Servers option:

This is a command line option that attempts to stop all running servers that uses the same installation directory as the CE. When this option is given on the command line like this:

```
ibmditk -tdishutdown
```

then the CE will start and look at every defined server in the Tivoli Directory Integrator servers project, filtering out those that do not use the same installation directory as the CE and attempt to stop it. When this is done the CE will exit the Java VM with an exit code of zero. There is no guarantee that the servers the CE tried to stop actually did stop. Some servers may linger beyond the time it takes the CE to complete this command and some servers may simply refuse to stop for various reasons.

Perspective option:

This is a command line option that instructs the CE to open in an alternative perspective. Currently, the only perspective other than the default one is the Easy ETL perspective, and the CE is started with it using the following option:

```
ibmditk -perspective com.ibm.tdi.rcp.perspective.etl
```

Server

The following command-line options are for the IBM Tivoli Directory Integrator server (`ibmdisrv [options]`):

Example:

```
ibmdisrv -c"C:\demos\rs.xml" -r"Access2LDAP" -l"c:\metamerge\mydemo.log"
```

Notes:

1. There is no space between the option letter and the value. Use quotes to save against possible spaces or commas in the values.
2. The Windows Shell executive allows a maximum of nine (9) arguments, from the list below. There aren't any limitations on other platforms.
3. Do not use comma (,) in the configuration file name.

-s <dir>

Specifies the working directory where the solution is located; this directory is known as the Solution Directory. All relative file references in Tivoli Directory Integrator and in your Configs and so forth will be relative to this location. Must be the first parameter specified.

If the specified directory does not exists, it will be created by the Tivoli Directory Integrator server, and populated with a number of properties files (based upon those in the installation directory) that you can tailor to your needs. See Appendix B, "Example Property files," on page 325 for more information.

-c <file...>

Configuration file(s). If you don't specify this option, the items in the Autostart folder will be loaded and started (unless suppressed by specifying **-D**). Wildcards, as in `*.xml`, are allowed too.

Note: Submitting multiple configuration files is only allowed if the **-d** option is also specified.

-n <encoding>

Encoding to be used to write Config files. This must be a valid character set identifier valid in Java2; refer to the IANA Charset Registry (<http://www.iana.org/assignments/character-sets>) for the full list of values. Note that Java2 only supports a subset of those.

-r <al...>

List of AssemblyLine names to start. To start AssemblyLine **a** and **b**, use the command **-r a b**. Other syntaxes are supported as well: **-ra,b**; **-ra -rb**.

Note: If you use includes and namespaces, the AssemblyLine can be myNamespace:/AssemblyLines/alName (assuming namespace **myNamespace** and AssemblyLine name **alName**).

-T<name>

Enable JLOG-style tracing to file `trace<name>.log`, in directory `<Tivoli_Common_Dir>/TDI/logs/`. Default is trace to memory (from which it can be retrieved by the traceback routines of JFFDC in case of an unhandled exception.)

-D Flag to disable startup of items in the Autostart folder.

-w If **-r** (or **-t**) is specified then this flag causes IBM Tivoli Directory Integrator to wait for each AssemblyLine to complete before starting the next. If this flag is not specified then IBM Tivoli Directory Integrator starts all AssemblyLines specified by the **-r** parameter in parallel. When the last AssemblyLine has finished, the server stops.

-e Specifying this option causes the server to run in Secure mode. Using the master password specific to this server, it will decrypt and encrypt all Config files as well as the server API Registry.

-v Show version information and exit. This is logged in the log file only.

-P <password>

Password if configuration file(s) is/are encrypted.

-p Dump Java properties on startup. Note that you still must provide a configuration file, which is read before Java properties are dumped.

-d Start a "daemon", or *Config Instance* on this system.

If you start with -d, you will start one anonymous instance (the daemon), which will start one config instance for each config file specified on the command line; this allows you to start multiple config instances on the fly. You may specify 0 or more config files on the command line. It does not make sense to specify any AssemblyLines to run in this mode, since it is impossible to state which config file the AL will be in. You can autostart AssemblyLines, though, since those belong to the config instance that specifies the autostart.

If you start without -d, you will get one config instance that loads the config file specified on the command line. You must specify exactly one config file on the command line. (If you must use multiple config files, they may be piped in on standard input.) In this mode, you can specify any number of AssemblyLines to run. This is the traditional way of running the server.

-q Takes 1 argument, mode. Mode=1 means run in record mode, mode=2 means run in playback mode.

-l <file>

Log file (default console output). Does very little as few messages go to the console. To change the log file for most of the logging, change `log4j.properties`.

-R Disables the Remote API, regardless of the setting in `global.properties`.

-W Start all Configs in the same thread; they do not terminate but wait forever.

-M Start AssemblyLines in simulation mode.

-S This option is for internal use for communication between the Configuration Editor and a server only; it is used to pass Config Files between them. Do not use this option yourself.

-f extProp1=file1, extProp2=file2

Where `extProp` is the name of the external Property Store. `file` specifies from where to read the properties. This option specifies a user-defined, external Property Store that can be entered when starting a TDI server. This optional command-line parameter `-f` can be used with the "ibmdisrv" server startup scripts. `extProp` is the name of the external Property Store. `file` specifies from where to read the properties. When the `-f` option is used to specify a properties file from the command line, the server changes the Property Store configuration in memory only, i.e. the server does not make this change permanent by changing the TDI Config file on disk – this change is valid for the current run of the TDI server.

If any property files are specified at the command line, they are valid only for the Config Instances specified with the `-c` command-line option (which are loaded on TDI server startup). The property files specified at the command line do not have any impact on Config Instances which have not been explicitly named with the `-c` command-line option (these can be Config Instances loaded by remote server API client for example).

If a Property Store whose name is specified with the `-f` command-line switch cannot be found in a Config Instance, an error message is logged in the server log (`ibmdi.log` in the Install-directory). When a Property Store name is specified more than once with the `-f` command-line switch then there are two effects: (1) a warning message is logged, and (2) the file specified last will take effect. This feature is implemented in the `com.ibm.di.server.RS` Java class (referenced by way of the main variable when scripting). After the `reload()` method is called, the `MetamergeConfig` object is loaded, and for each Property Store specified on the command line, the corresponding `PropertyStoreConfig` object is updated.

Note:

Although Copy/Paste of Config objects (ALs, Connectors, FCs, and so forth) are fully supported. You can easily copy ALs and components and then paste them into another Config. You can also exchange ALs and components using IM chats, e-mails and text files, because the copy-buffer is filled with the TDI Config XML definition of the selected item. This makes passing stuff around simple and easy, and is a great tool for support and online assistance (for example, ICT/NotesBuddy, forums, ...).

Note:

Make sure you select the entire <MetamergeConfig> node in your copy command, including the start and end tags.

-i This option specifies that the TDI server ignores any properties from the `global.properties` file, and reads only the `solution.properties` file. This option can be used when the `global.properties` file is unreadable - for example, when the encoding the TDI server is started with is different from the encoding of `global.properties`.

-? Prints a *usage* message, showing all options in brief.

-j <file>

This option is used to read regression information from the specified file, and is compared with the details produced by the AssemblyLine. If there are variations, warning messages are written to the log file. This option is useful only when running a single AssemblyLine.

-J<file>

This option is used to write the AssemblyLine regression information to the specified file.

-k<file>

This option is used to ignore the work Entry when reading the regression information.

Notes:

1. If a property is used as a parameter for a Connector, Parser, or Function Component, and that property does not exist in the property store, a warning message is logged.
2. You can load a configuration file into the server without starting the AssemblyLines. Warning messages are logged for the non-existing properties that are used.

When IBM Tivoli Directory Integrator ends it returns one of the following exit codes:

0 No error. The operation has been successful.

1

- Cannot open log file (-l parameter)
- Cannot open Config file
- An AssemblyLine failed. Applicable only if the Server is run in non-daemon mode, that is, without the "-d" option. For example: "ibmdirsv -c rs.xml -r al" or "ibmdirsv -c rs.xml -r al1,al2,al3".

2 (Obsolete) Exit after auto-run. When you start IBM Tivoli Directory Integrator specifying -w, the Server runs the AssemblyLines specified by the -r parameter and then exits.

Note: AssemblyLines run from the Configuration Editor are started in a different way and will not exit with status 2.

9 (Obsolete) License expired or invalid.

Note: If the Server is shutdown by an administration request and a custom exit code is specified, that custom code will be used as the exit code of the Server.

Command Line Interface – *tdisrvctl* utility

The Command Line Interface (CLI) to Tivoli Directory Integrator, called the *tdisrvctl* utility, is designed for remotely managing Configs, AssemblyLines, and so on. This utility connects to a remote Tivoli Directory Integrator server using the 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 6, “Security and TDI,” on page 89.

It exposes various command line options for the following functions:

- Start, stop, or reload Tivoli Directory Integrator Configs.
- Start or stop AssemblyLines in a particular config.
- Display a list of configs loaded on the server.
- Shutdown server.
- Display config report.
- Manage config properties through TDI-p, the Tivoli Directory Integrator properties framework
- Send custom notification events.
- View exposed AL Operations.
- View tombstones for terminated Configs and AssemblyLines
- View Tivoli Directory Integrator Server details.

Notes:

1. The command line utility is shipped in the *TDI_install_dir/bin* folder.
2. Remote Method Invocation (RMI) is enabled by default. Therefore, for any remote server API client (including the CLI), the property *api.remote.on* should be set to *true* and the IP address of the client must be mentioned in the property *api.remote.nonssl.hosts* in the *global.properties* (or *solution.properties*) file of the remote Tivoli Directory Integrator Server (if non-SSL mode is being used).
3. The remote Tivoli Directory Integrator server must be running.
4. Do not use comma (,) in the configuration file name.

Command Line Reference

The command has the following usage:

tdisrvctl [general_options] **-op** operation [operation_specific_options]

where **general_options** can be:

-h host	Enter the remote server IP address or hostname (default is localhost).
-K keystore	Enter the name of the SSL key database file.
-p port	Enter the port number (default is 1099).
-P key_pwd	Enter the key file password.
-s	Specify the working directory where the solution directory is located.
-T truststore	Enter the name of the SSL truststore database file.
-u userID	Enter the username (for custom authentication).
-v	Run in verbose mode.
-w user_pwd	Enter the user password (for custom authentication).
-W trust_pwd	Enter the trust file password.
-?	Display command usage.

And **operation** can be:

event	Send custom notification events
prop	Manage Config properties
queryop	Query for AssemblyLine (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
start	Start specific Config or ALs
stop	Stop specific Config or ALs
tombstone	View tombstone entries for specific Config or AL.
deletetombstone	delete a tombstone entry
debug	debug components of a running AssemblyLine

You can display help for any particular option like this:

```
tdisrvctl -op operation -?
```

Operations

- event** Use this option to send custom notification events to a particular server. All listeners registered for the particular event receive this notification. This allows 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 are 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	Name the config to work with.
-l	List all the property stores configured.
-o property_store	Name 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' options are mutually exclusive, and cannot be used together.
2. The '-e' option can only be used with the '-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.

This option is useful in a scripting environment. A Tivoli Directory Integrator solution developer can develop a script to automatically query for exposed operations and then use the result to start an AssemblyLine with a specific operation using the start operation's **-r -alop** flag. The output of this operation is such that it can be grepped for or tokenized easily in a scripted environment.

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

Output:

```
ALOp:{attr_1;attr_2...attr_n;}
```

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 ADAAssemblyLine
```

Example Output:

```
$initialize: {ldapurl;loginPasswd;loginUserName}
```

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.jks -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 lists details of the particular config. The details are AssemblyLines, Connectors and Parsers in each AssemblyLine, Connector library, Parser library, Script library, Function Library. This option gives a one shot view of 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 that 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 the 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.

The displayed details for each connector or function component part of an AssemblyLine look like this:

Name	:	count
Mode	:	Iterator
State	:	Enabled
Debug	:	Disabled
Template	:	system:/Connectors/ibmdi.Timer
Parser	:	[parent]
Comment	:	None

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.
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 Tivoli Directory Integrator server.

The format for this command is:

```
tdisrvctl [general_options] -op shutdown [-o return_code] [-f]
```

where:

-o return_code	The return code with which the remote Tivoli Directory Integrator server should exit.
-f	Force a controlled shutdown and exit all AssemblyLines.

Examples:

To shutdown the local Tivoli Directory Integrator server:

```
tdisrvctl -op shutdown
```

To shutdown the local Tivoli Directory Integrator server, with a controlled shutdown of all AssemblyLines:

```
tdisrvctl -op shutdown -f
```

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 Tivoli Directory Integrator server.

The usage of the command is:

```
tdisrvctl [general_options] -op srvinfo
```

Example:

To view the server information for a Tivoli Directory Integrator server running on localhost

```
tdisrvctl -h localhost -op srvInfo
```

status This option can be used to view status of AssemblyLines.

The usage for status operation is:

```
tdisrvctl [general_options] -op status -c [config_list | all]
          -r [AL_list | all]
          -listen
```

where:

config_list	Comma-separated list of Configs or keyword "all".
--------------------	---

AL_list	Comma-separated list of ALs or keyword "all".
-listen	indicates to start receiving the logs of a running Config or AssemblyLine.

Notes:

1. At least one of the options ('-c' or '-r') must be specified. -
2. The keyword "all" indicates all configs or AssemblyLines.
3. The -listen option requires exactly one Config or AssemblyLine to be specified.
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 the status of all configs and ALs:

```
tdisrvctl [general_options] -op status -c all -r all
```

You can also write

```
tdisrvctl [general_options] -op status
```

To see the status of AL1, AL2:

```
tdisrvctl -h itditest -op status -c c1.xml -r AL1,AL2
```

Output:

```
(Component Type # Component Name # RUNNING / STOPPED # Statistics):
1 # AL1 # RUNNING # [get:571] [add:571] [del:3] [requests:2333]....
1 # AL2 # STOPPED #
```

The Component Types are:

- 0 for Config
- 1 for Assembly line

The Statistics contain the following details (valid for AssemblyLines 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
```

To see the details of a running AssemblyLine on a particular server and start receiving its logs:

```
tdisrvctl -h itditest -op status -c rs.xml -r all -listen
```

start This option can be used to start a config or AssemblyLines.

The usage for the start operation is:

```
tdisrvctl [general_options] -op start -c [config]
  -e [password]
    -r [AL_list | all] -alop <alop_Name> [{requiredAttr_1; requiredAttr_2; ...}
      requiredAttr_n] | [-f filename]
  -s [Simulate mode]
  -m [run name] -o [propStore1=filename1,propStore2=filename2...]
  -t [temp config instance]
  -listen
  -sync
```

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'.
-o property file list	comma separated list of property store names and values
-alop operName	The specific AL operation and list of list required attributes for the specified operation.
-f filename	Name of the file where the input attributes and their values are configured for the operation.
-s Simulate mode	Run the specified AssemblyLines in simulate mode
-m multi-instance	Run multiple instances of same Config with different run names
-t temp config instance	Start temp config instance from the XML in the config file specified
-listen	receive the logs of the specified Config or AssemblyLine
-sync	execute AssemblyLine synchronously

Notes:

1. The '-c' option is mandatory. -
2. The keyword "all" indicates all AssemblyLines.
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. When running a temp config with solution or run name it is not possible to check if another config with the same name is already running on the server. If this happens an exception will occur. You could check the running config instances using the **status** command.
6. The -t option expects the Config specified in the -c option to be located on the client machine.
7. If the -t option is used and the config specified in the -c option is relative then it will be searched in the current folder.
8. The -listen option requires exactly one Config or AssemblyLine to be specified.
9. The -listen option executes an AssemblyLine synchronously. There is no need to combine it with the -sync option.
10. The -sync option requires exactly one AssemblyLine to be specified.

Examples:

1. To start assembly line AL1 and AL2 of config C1 on remote server itditest:

```
tdisrvctl -h itditest -T trust.kdb -W secret -op start -c C1.xml -r AL1,AL2
```

The **-r** option requires that **-c** option should also be specified. This is because the AssemblyLines 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 $initialize {ldapurl:ldap://9.182.190.149:390;loginPasswd:password;loginUsername:cn=root}
```

3. To start AssemblyLine AL1 on remote server itditest with AL operation update:

```
tdisrvctl -h itditest -T trust.kdb -W secret -op start  
-c examples/ADCustomConnector.xml -r ADAssemblyLine  
-alop search {$init.ldapurl:ldap://9.182.190.149:390;$init.loginPasswd:password;$init.loginUsername:cn=root;searchBase:o=ibm,c=us}
```

Note: All initialization attributes are to be prefixed with \$init.

- 4.

```
tdisrvctl -h itditest -T trust.kdb -W secret -op start -c examples/ADCustomConnector.xml -r ADAssemblyLine -alop search -f inputFile
```

Input file format:

=====

Key1:value1

Key2:value2

5. Command to run an AssemblyLine AL1 in simulate mode:

```
tdisrvctl -h itditest -T trust.kdb -W secret -op start -c examples/ADCustomConnector.xml -r AL1 -s
```

6. Command to load multiple config instances:

```
tdisrvctl -op start -c C1.xml -m test -f PropertyStorename=TestProp.properties, PropStore2=propfile2 ... -r AL1,AL2
```

7. Command to run temp config instance:

```
tdisrvctl -op start -c C1.xml -t -r AL1
```

8. Command to start a config on a particular server and receive its logs:

```
tdisrvctl -h itditest -op start -c rs.xml -listen
```

9. Command to start an AssemblyLine on a particular server and receive its logs:

```
tdisrvctl -h itditest -op start -c rs.xml -r AL1 -listen
```

10. Command to execute an assembly line synchronously on a particular server:

```
tdisrvctl -h itditest -op start -c rs.xml -r AL1 -sync
```

stop The usage for the stop operation is:

```
tdisrvctl [general_options] -op stop -c [config]  
-r [AL_list | all]
```

where:

-c config	Name of Config.
-r AL_list	Comma-separated list of ALs to stop or keyword "all."
-f	Force a controlled shutdown of AssemblyLines.

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.

4. The **-r** option requires that **-c** option should also be specified. This is because the AssemblyLines mentioned in the command *must* belong to one of the Configs in the **-c** option.
5. The **-f** option is optional.
6. 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, of Config C1 on remote server itditest:

```
tdisrvctl -h itditest -T trust.jks -W secret -op stop -c C1.xml -r AL1,AL2
```

tombstone

This option can be used to view tombstone details of previously run Configs, AssemblyLines and EventHandlers (historical).

The usage for the tombstone operation is:

```
tdisrvctl [general_options] -op tombstone -c [config]
           [-r [AL_name] ]
           [-age n]
           [[attribute_list] | all ]
```

where:

-age n	Tombstone record for the last 'n' days (default is 1 day).
-c config	Name of Config.
-r AL_name	Name of AssemblyLine.
all	Tombstone attributes: show all.

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. 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
```

deletetombstone

This option can be used to delete tombstone entries for previously run AssemblyLines.

The usage for the delete tombstone operation is:

```
tdisrvctl [general_options] -op deletetombstone -guid <GUID number>
```

where

-guid "GUID number" is the unique identifier for the tombstone to be deleted. The GUID for a tombstone can be obtained by viewing the contents of the tombstone; see the entry about the tombstone option for details as to how to obtain the GUID.

- debug** This option can be used to set the Debug mode values of connectors and function components of a running AssemblyLine. When you set the Debug mode of a connector with specified parser, the Debug mode of the parser is also initialized with the same value.

The usage for the debug operation is as follows:

```
tdisrvctl [general_options] -op debug -c config
          -r assembly_line
          [-alc al_component]
          -on/off
```

where:

-c config	Name of Config.
-r assembly_line	Name of AssemblyLine.
-alc al_component	name of the AssemblyLine component.
-on	flag to enable debug.
-off	flag to disable debug.

Notes:

1. The '-c' and '-r' options are mandatory and require exactly one Config/AssemblyLine to be specified.
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 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".
4. If the -alc option is not specified all components in the specified AssemblyLine will be affected.

Examples:

1. To view the Debug mode value of the components in the AssemblyLines of a specified Config:

```
tdisrvctl -op report -c C1
```
2. To enable the debug mode for all components in the running AssemblyLine al2:

```
tdisrvctl -op debug -c C1-r al2 -on C1-r al2 -on
```
3. To disable the debug mode for specified components in the running AssemblyLine al3:

```
tdisrvctl -op debug -c C1-r al3 -alc comp1,comp2 -off
```

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 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 **tdisrvctl** command will return an exit code of zero if the operation is successful and non-zero if the operation fails. Possible reasons for operation failure are:
 - A connection cannot be established to the remote Server.
 - The remote Server reported an error (probably related to the operation that was executed).
 - An AssemblyLine, which is executed synchronously, failed (see the "-sync" option of the "start" operation).
- 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. The command uses a file called **tdisrvctl-log4j.properties** to set up the Log4J logging. If the solution directory is specified the command sets an environment variable for pointing to the log configuration file in the solution directory. If the solution directory is not specified then the command uses the log configuration file present in the install directory.
- The **tdisrvctl-log4j.properties** file has the complete path of location where the logs are to be created. The log files are created in the *TDI_install_dir/logs* directory by default. The location can be customized as needed.
- All reported error and warning messages are displayed with an error code prefix. This error code can be used to search the *IBM Tivoli Directory Integrator V7.1.1 Messages Guide* for an explanation of the error message and operator response.

Chapter 14. Logging and debugging

Tivoli Directory Integrator uses a logging class to record messages to a number of various log channels. All Tivoli Directory Integrator components use this logging class which in turn invokes an industry standard logging tool (Log4J). While Log4J provides a variety of output channels and formats, there are other logging utilities with overlapping and additional output channels that you as a Tivoli Directory Integrator user may need. Many of these are open source libraries that are not bundled with Tivoli Directory Integrator for legal reasons. To enable inclusion of these 3rd party logging utilities, the Tivoli Directory Integrator logging component is modeled to act as a proxy between Tivoli Directory Integrator and the actual logging implementations, called LogInterface implementations. Refer to the section "Creating additional Loggers" in *IBM Tivoli Directory Integrator V7.1.1 Reference Guide* for more information on how to create, configure and program your own LogInterface classes.

Note: Enable or disable logging in Tivoli Directory Integrator by configuring the `com.ibm.di.logging.enabled` property. To enable logging, use `com.ibm.di.logging.enabled=true` (default). To completely disable logging, use `com.ibm.di.logging.enabled=false`.

The remainder of this section describes how to use the logging class that is bundled with Tivoli Directory Integrator, called `com.ibm.di.log.TDILog4J`.

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.

The Log4J logging engine 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. By means of the aforementioned logging class, Tivoli Directory Integrator has additional tracing facilities (discussed in Chapter 15, "Tracing and FFDC," on page 209), though in most cases, the logging functionality described here suffice.

Some Tivoli Directory Integrator components may have very specific troubleshooting guidelines; always check the particular component's section in the *IBM Tivoli Directory Integrator V7.1.1 Reference Guide* and the *IBM Tivoli Directory Integrator V7.1.1 Problem Determination Guide* for more information.

The log scheme for the server (ibmdisrv) is described by the file `Log4J.properties` and elements of the Config file, see "Log4J default parameters" on page 206.

Note: Any of the aforementioned properties files can be located in the Solution 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 <http://jakarta.apache.org/log4j/docs>.

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 V7.1.1 Users Guide*, in which you also find out how the interactive debugger works.

Script-based logging

You can issue messages to the AssemblyLine's configured loggers at any time using JavaScript, at any point where scripting is possible (hooks, script components, and so on.) The explicit `logmsg()` calls available to you (that is, `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.

If you use

```
task.logmsg()
```

your messages will be logged along with the other messages from the AssemblyLine. If you are running your AssemblyLine from the Configuration Editor, that will be in the CE output window. If your AssemblyLine also uses other logging methods, the messages will be there too.

When you use

```
main.logmsg()
```

your message will be logged along with other messages from the Config Instance. This will be in the log file(s) or other loggers created by the Config Instance, which are typically not seen in the Configuration Editor.

Logging using the default Log4J class

Configuring the default logging of IBM Tivoli Directory Integrator, which uses Apache Log4J is done globally (using the file `Log4J.properties` which specifies global defaults for Server tasks) or specifically, using the Configuration Editor, for each AssemblyLine 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` file; see "Log4J default parameters" on page 206.

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://download.oracle.com/javase/6/docs/technotes/guides/intl/encoding.doc.html>.

The possible log schemes are as follows:

FileRollerAppender

Sometimes, you want to log to file but keep a limited number of files, as they can fill your disks. FileRollerAppender 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.

FileRollerAppender 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 FileRollerAppender, 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 207.

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, and so on.

Log Enabled

Click to enable the use of this Appender.

ConsoleAppender

Logs to the console (standard output). This is in the window where you started the server (ibmdisrv) or the execute task-window in the Configuration Editor (ibmditk). **Console** has the following parameters:

Layout

See **FileRollerAppender**, previous.

Pattern

See **FileRollerAppender**, previous.

Log level

See **FileRollerAppender**, previous.

Log Enabled

See **FileRollerAppender**, previous.

FileAppender

Logs to a file. **File** has the following parameters:

File Path

See **FileRollerAppender**, previous.

Append to file

Click to append log information to file. If option is not enabled, the file is overwritten.

Layout

See [FileRollerAppender](#), previous.

Pattern

See [FileRollerAppender](#), previous.

Log level

See [FileRollerAppender](#), previous.

Character Encoding

Character Encoding to be used; like Cp1252, ISO-8859-1, and so on.

Log Enabled

See [FileRollerAppender](#), previous.

This is the appender set up by default; see “Log4J default parameters” on page 206.

SyslogAppender

Enables IBM Tivoli Directory Integrator to log on UNIX Syslog. **Syslog** has the following parameters:

Host name/IP Address

Host to log on 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 [FileRollerAppender](#), previous.

Pattern

See [FileRollerAppender](#), previous.

Log level

See [FileRollerAppender](#), previous.

Log Enabled

See [FileRollerAppender](#), previous.

NTEventLog

Enables applications to log to the Windows NT Event Log (on Windows platforms). **NTEventLog** has the following parameters:

Source

The "source" name appearing in the NT event log; usually the title of the application doing the logging.

Layout

See [FileRollerAppender](#), previous.

Pattern

See [FileRollerAppender](#), previous.

Log level

See [FileRollerAppender](#), previous.

Log Enabled

See [FileRollerAppender](#), previous.

DailyRollingFileAppender

The daily rolling file appender rotates the log file every day. When the output file is rolled it is given a name consisting of the base name plus a date pattern string; that is, filename.yyyy-mm-dd. It usually is used with the **Append to file** parameter set to **true**. **DailyRollingFile** has the following parameters:

File Path

See **FileRollerAppender**, 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 **FileRollerAppender**, previous.

Pattern

See **FileRollerAppender**, previous.

Log level

See **FileRollerAppender**, previous.

Character Encoding

Character Encoding to be used; like Cp1252, ISO-8859-1, and so on.

Log Enabled

See **FileRollerAppender**, previous.

Also see the example under "Log4J default parameters" on page 206.

SystemLogAppender

This Appender creates log files in a catalog hierarchy under *TDI_install_dir/system_logs*. For each Config File, there is a corresponding directory with logfiles named *AL_xxx*, where *xxx* is the name of the AssemblyLine 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 **FileRollerAppender**, previous.

Character Encoding

Character Encoding to be used; like Cp1252, ISO-8859-1, and so on.

Log Enabled

See **FileRollerAppender**, 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 Tivoli Directory Integrator allows scripting. It can take one or two parameters. See the Java API documentation 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

When Tivoli Directory Integrator is installed, a *FileAppender* is used for the default logger. If you want to change the default logger you must change the content of the *log4j.properties* file situated in the *TDI_installdir/etc* folder. The default configuration is as follows:

```
# 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=logs/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
```

The FileAppender logger truncates the content of the *ibmdi.log* file (situated in *TDI_installdir/logs*) each time the Tivoli Directory Integrator server is started. If want to change that behavior you must change the **log4j.appender.Default.append** property to true.

In the *log4j.properties* file you can find also two examples for changing the default logger to RollingFileAppender or DailyRollingFileAppender. If you want to use them just uncomment the preferred one and comment the FileAppender logger:

```
#####
# ROLLING FILE SIZE APPENDER
##RollingFileAppender rolls over log files when they reach a certain size specified by the
##MaxFileSize parameter
```

```
#log4j.appender.Default=org.apache.log4j.RollingFileAppender
#log4j.appender.Default.File=logs/ibmdi.log
#log4j.appender.Default.Append=true
#log4j.appender.Default.MaxFileSize=10MB
```

```

#log4j.appender.Default.MaxBackupIndex=10
#log4j.appender.Default.layout=org.apache.log4j.PatternLayout
#log4j.appender.Default.layout.ConversionPattern=%d{ISO8601} %-5p [%c] - %m%n

#####DAILY OUTPUT LOG4J SETTINGS
## With the DailyRollingFileAppender the underlying file is rolled over at a user chosen frequency.
##The rolling schedule is specified by the DatePattern option

#log4j.appender.Default=org.apache.log4j.DailyRollingFileAppender
#log4j.appender.Default.file=logs/ibmdi.log
#log4j.appender.Default.DatePattern='yyyy-MM-dd
#log4j.appender.Default.layout=org.apache.log4j.PatternLayout
#log4j.appender.Default.layout.ConversionPattern=%d{ISO8601} %-5p [%c] - %m%n

```

These are some of the parameters you find in the file Log4J.properties (for ibmdisrv and ibmditk).

Full documentation can be found at <http://jakarta.apache.org/log4j/docs>.

Log4J.rootCategory=DEBUG, Default

DEBUG is the loglevel for the named Appender (Log4J term called Default). If you set the loglevel to OFF or to the level above INFO, you do not get output from your script logmessages (see the following log terms):

Log4J.appenders.Default

Defines what type of Appender the named appender Default is. It can be one of the following:

- FileRollerAppender (generates a new file for each run of the Server)
- ConsoleAppender (log to console)
- FileAppender (log to file)
- SyslogAppender (log to UNIX Syslog)
- NTEventLog (log to Windows NT EventLog)
- DailyRollingFileAppender (saves old files with a timestamp in their names)
- SystemLogAppender (In a folder structure under *root_directory/system_logs*)

Log4J.appenders.Default.file

Default log file for FileAppender, 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 Configuration 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 log.

Note: This information is intended for users who want to continue using the *global.properties* file to customize logging output. You can customize logging output through the Configuration Editor (ibmditk).

The following section defines a log scheme called CONSOLE, which later can be used by specific AssemblyLines:

```

Log4J.appenders.CONSOLE=org.apache.Log4J.ConsoleAppender
Log4J.appenders.CONSOLE.layout=org.apache.Log4J.PatternLayout
Log4J.appenders.CONSOLE.layout.ConversionPattern=%d [%t] %-5p - %m%n

```

Now in order to have the AssemblyLines myAL use this, you need the lines:

```
Log4J.logger.AssemblyLine.myAL=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 named by the creator), and *xxx* is a unique ID for the thread. **%c{2}** outputs *alName & unique ID*.

%m Message.

%n Newline.

%t Threadname.

Chapter 15. Tracing and FFDC

In addition to the user-configurable logging functionality described in Chapter 14, “Logging and debugging,” on page 201, 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 14, “Logging and debugging,” on page 201. 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.

Tracing Enhancements

Currently most Connectors and Parsers have entry and exit trace statements. For 7.1.1, a number of classes on the TDI server have trace statements added to:

- Method entry and exit points.
- Interactions with third party software.
- Thread creations.

Understanding Tracing

Tracing is done in the code of Tivoli Directory Integrator using JLOG's PDLogger object. PDLogger or the Problem Determination 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]

Basic tracing information means Time, Level (Min, Mid, Max), Location in code, that is Method name and Entry/Exit. 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 (that is, an 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 prefix: **CTGDI**.

All error messages are prefixed with a unique TMSXML messageID that indicates the cause of the error and an operator response.

All info messages are also prefixed with a unique TMSXML messageID that 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 level (hierarchy, from most severe to least severe):

- FATAL
- ERROR
- WARNING
- INFO
- DEBUG_MIN
- DEBUG_MID
- DEBUG_MAX

The default level is FATAL.

Default Trace level as well as whether Tracing is done to file or memory is defined in the default `jlog.properties` file. This file is placed in the `TDI_install_dir/etc` folder. If you use a solution directory, it is placed in the `TDI_Solution_dir/etc` folder.

Setting trace levels dynamically

Tivoli Directory Integrator ships a `LogCmd.bat` (for Windows) and `LogCmd.sh` (for Unixes) scripts. By using them the Trace properties can be set dynamically. JLOG logger starts a command server on the default port (9992) to listen for log commands sent by the `logcmd` command line utility.

For the `logcmd` scripts to work, the command server needs to be started first. To start the log command server, you need to set `jlog.noLogCmd=false` in the `jlog.properties` file.

The listen port of this server can be changed by setting the `jlog.logCmdPort` property in the `jlog.properties` file to the desired value. For more information about these properties read the comments in the `jlog.properties` file.

Usage of the `logcmd` command is as follows:

```
logcmd -o port_number { [-h] | [help] |
    [list {node_name} ] |
    [config node_name] |
    [set node_name key_name=value] |
    [remove node_name {key_name} ] |
    [dump handler_name] | [save {all}] }
```

where

-o port_number

The port number to use to connect to the log command server. If not specified the default port (9992) is assumed.

-h | help

Displays syntax information for the command.

list Lists the names of all known logging objects (nodes).

list node_name

Lists the names of the children of the node name. Not all logging objects have children.

config node_name

Lists the all the configuration properties for the node.

set node_name key_name=value

Sets a property key for the node name. If the logging object, node_name, does not exist, the logging object is created and the property is added.

remove node_name

Removes the configuration object node_name. A logging object that has been instantiated from this configuration is not affected by removing the configuration node.

remove node_name key_name

Removes the configuration property key_name from the logging object node_name. If the object supports a hierarchical inheritance of properties, a subsequent logcmd config node_name command may show the key just removed. In that case, it was inherited from an ancestor.

save {all}

Saves the logging configuration to persistent store. If **all** is specified, the entire configuration is saved; otherwise, only those configuration nodes that were originally loaded from the file are saved.

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	The handler immediately sends all the queued events to its output listeners when the property is set to true. The property can 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	The log events are dumped to the snap shot file synchronously when the property is set to true. 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.levelflit	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.	The listed IDs are blocked when the msgIDs property is set to BLOCK. When set to PASSTHRU, the listed IDs are sent to the filter.
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.levelflit. 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.levelflit.className=com.ibm.log.LevelFilter
jlog.levelflit.level=FATAL
```

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.levelflit.level property can take any of the other Log level values as well, but only values of ERROR or FATAL make much sense as otherwise the amount of FFDC dumping is very high, causing huge slowdowns of the TDI Server.

Chapter 16. Administration and Monitoring

The IBM Tivoli Directory Integrator (TDI) 7.1.1 Administration and Monitoring Console (AMC) user interface is deployed into the Integrated Solutions Console (ISC). Use the AMC to start, stop and manage Tivoli Directory Integrator Configs and AssemblyLines remotely.

IBM Tivoli Directory Integrator 7.1.1 also ships an Action Manager with the AMC. The Action Manager is a stand-alone Java application that interacts with the AMC database and uses the Remote Server API to manage remote AssemblyLines.

The Administration and Monitoring Console is comprised of a Java WAR file (Web Archive) and a WAB file (Web Bundle) that can be deployed on ISC SE and Tivoli Integrated Portal (ISC embedded).

The current Action Manager, bundled with IBM Tivoli Directory Integrator 7.1.1 AMC, supports Tivoli Directory Integrator 7.1.1, TDI 6.1.X and TDI 6.0. Note that the Action Manager for IBM Tivoli Directory Integrator 7.1.1 supports TDI 6.1.X and TDI 6.0 with some restrictions. Versions of Tivoli Directory Integrator prior to version 6.0 are not supported.

Note: IBM Tivoli Directory Integrator 7.1.1 and solutions developed and deployed with it can also be monitored with IBM Tivoli Monitoring (ITM) Server and Portal or IBM Netcool®/OMNIBUS, by virtue of Tivoli Directory Integrator's Java Management Extension (JMX) interface. You can find supported examples that show how you can accomplish this in the appendix entitled, Appendix C, "Monitoring with external tools," on page 335.

Installation and Configuration

See "Installing IBM Tivoli Directory Integrator" on page 8 for information about installing Tivoli Directory Integrator and the Administration and Monitoring Console. Installing AMC also installs the Action Manager. If you choose a custom Tivoli Integrated Portal (ISC embedded) to deploy AMC or defer deployment of AMC during installation, see "Deploying AMC to a custom ISC SE or Tivoli Integrated Portal (ISC embedded)" on page 47 for information on additional deployment requirements.

Deploying AMC into the Integrated Solutions Console (ISC)

These instructions require that you be familiar with the IBM Tivoli Directory Integrator 7.1.1 Installation procedures. See "Using the platform-specific TDI installer" on page 11 for information about Tivoli Directory Integrator Installation. Installing AMC also installs the Action Manager.

If you want to deploy Tivoli Directory Integrator Administration and Monitoring Console into the ISC automatically, select one of the following options:

- Embedded instance of ISC SE
- Existing instance of ISC

If you do not want to deploy AMC into ISC automatically at installation time, select "Do not specify. I will manually deploy AMC at a later time."

The installer automatically installs ISC and deploys AMC in it if you select "Embedded instance of ISC SE" or "Existing instance of ISC" during IBM Tivoli Directory Integrator 7.1.1 installation.

To install and deploy the Administration and Monitoring Console into the ISC SE:

1. Invoke the Tivoli Directory Integrator 7.1.1 installer.
2. During installation, select the **Custom** installation. (Typical installation does not offer the AMC option.)

3. On the "Select Features" window of the installation, select **AMC: Administration Monitoring Console** and **embedded Web platform** (includes Integrated Solutions Console, Standard Edition (ISC SE)).
4. Finish installing Tivoli Directory Integrator 7.1.1.

Deploying AMC as a Windows service or UNIX process using the TDI installer

You can register AMC as a Windows service or UNIX process if the following conditions are satisfied:

Note: You cannot register AMC as a service on i5/OS.

- The person installing Tivoli Directory Integrator must have administrative permissions (Administrators group on Windows or root on UNIX).
- You have selected to install AMC into an "Embedded instance of ISC Standard Edition."
- You have selected "Register AMC as a system service" and given the service a name. The default service name is tdiamc

Deploying AMC on existing WAS environment

To deploy AMC on the existing WAS environment, modify the `tdiISCHome.bat` or `tdiISCHome.sh` script file to set the following parameters:

- Set the `TDI_ISC_RUNTIME` parameter to WAS
- Set the `TDI_ISC_HOME` parameter to WAS_HOME directory

For example:

```
set TDI_ISC_RUNTIME=WAS
set TDI_ISC_HOME=C:\Program Files\IBM\WebSphere\AppServer
```

Starting the Administration and Monitoring Console and Action Manager and logging in

You can start and stop the Administration and Monitoring Console and Action Manager by running the following scripts shipped in the `TDI_install_dir/bin/amc` folder:

- To start AMC, run the `start_tdiamc` script.
- To start AM, run the `startAM` script.

For more information about these scripts, see "AMC and AM Command line utilities" on page 251.

The above will start AMC and AM using a Derby database configured locally on the machine running in network mode on localhost at port 1528. For more information on alternate settings and configurations for both AMC and AM, see sections "Enabling AMC" on page 215 and "Enabling Action Manager" on page 223.

Once the Administration and Monitoring Console is started, you can access it from the following URL: `http://localhost:13100/ibm/console`; for more information, see "Log in and logout of the console" on page 229.

Stopping AMC and AM can be accomplished by running the `stop_tdiamc` and `stopAM` scripts, respectively.

Notes:

1. For information on adding users and user roles, see section "Console user authority" on page 218.
2. For information on AM, see section "Action Manager" on page 219.
3. For information on usage of individual panels in AMC, see the online panel help, or section "Administration and Monitoring Console User Interface" on page 229.
4. You will be registering Tivoli Directory Integrator Servers in AMC so that solutions from Tivoli Directory Integrator configurations can be administered. AMC will only be able to find Configs that each Tivoli Directory Integrator server has loaded when its started up. By default, Tivoli Directory Integrator 7.1.1 Servers have the remote server API enabled. Ensure that your `TDI_install_dir/`

- configs folder has the Configs you want to administer and monitor (or put them in the config folder of your solution directory if your server is using a solution directory).
- For an example walk through of on using AMC and Action Manager for a few simple tasks, see "Example walkthrough of creating a Solution View and Rules" on page 256.

Enabling AMC

The configuration file for the Administration and Monitoring Console is the `amc.properties` file that is located 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.

By default, the Administration and Monitoring Console makes use of Derby version 10 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 Derby database can be accessed in either the network mode or embedded mode. By default, AMC is shipped with Derby configured in network mode. The following properties in `amc.properties` are applicable to Derby configured for network mode:

```
com.ibm.di.amc.jdbc.database=jdbc:derby://localhost:1528/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=1528
com.ibm.di.amc.jdbc.sysibm=true
```

The property `com.ibm.di.amc.jdbc.database` points to Derby in network mode, running on `localhost:1528`. The database name being accessed is `tdiamcdb`, and `create=true`, indicating that AMC should create the database if not found.

You should change the `create=true` to `create=false` once your environment is set, so that in case the database path gets modified, AMC does not re-create the database, but instead throws a "Database not found" exception. You should also ensure that the database path be set to an absolute path to avoid any confusion about the database path later.

Other databases than Derby can be configured by setting the appropriate properties; see "JDBC Properties" on page 235.

AMC provides a separate startup and shutdown script for Action Manager. AMC allows the Action Manager to run remotely and provides a separate Derby start or shutdown script.

The Administration and Monitoring Console can also be configured to connect to the Derby database in Embedded Mode. In this case, the Action Manager, a separate application that also talks to the AMC database, is 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 Derby database. The following example shows the `amc.properties` file with Derby configured for embedded mode:

```
##Location of the database (embedded mode)
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.

Running Action Manager remotely

Beginning with Tivoli Directory Integrator 7.0, you can run Action Manager remotely without starting the AMC first. The database for AMC, **Derby**, must be running in Network mode in order for Action Manager to connect to it. Tivoli Directory Integrator 7.0 also provides start and shutdown scripts for the Derby data store so that a user can start Action Manager remotely without starting the AMC.

Notes:

1. Before you start Action Manager for the first time, you must have run AMC at least once. This is because AMC creates the necessary database tables required for AM.
2. You can find the scripts in this section in the following folder of the Install Directory of the remote computer: *TDI_install_dir\bin\amc\ActionManager*.
3. The instructions in the startup and shutdown sections that follow are for Action Manager and Derby running on different remote computers, and with AMC not running.
4. To verify that AMC, Action Manager or Derby has stopped, check the logs.

AMC and Action Manager startup: If you want to run Action Manager and Derby with AMC running, start AMC by typing `start_tdiamc.bat(sh)` and start the Action Manager by typing `startAM.bat(sh)`. The `tidamc` script calls the `startNetworkServer.bat(sh)` script, thereby starting the Derby database in network mode.

Note: The `startAM.bat(sh)` script has the Classpath defined for all the jars required by the Action Manager. There are two variables namely `CLASSPATH` and `DB_CLASSPATH`. The `DB_CLASSPATH` has the path separated list of JAR files required for achieving JDBC Connectivity with the database. When AMC is configured to use Oracle, MS SQL Server or DB2 the corresponding JDBC JAR files of these databases should be added to the `DB_CLASSPATH` variable.

AMC and Derby shutdown: The `stop_tdiamc.bat(sh)` script calls the `stopNetworkServer.bat(sh)` script. This ensures that the Derby Network server is stopped when AMC is shutdown.

Note: If Action Manager (AM) is running, this should be shut down first.

Action Manager remote startup: This section assumes that Action Manager and Derby are running on different computers.

1. Start Derby using the script `startNetworkServer.bat(sh)`.
2. Start Action Manager using the script `startAM.bat(sh)`.

The `startNetworkServer` script is used for starting the Derby database server in Network mode. The Derby server starts in Network mode on port 1528. The port selected is different from the default port for Derby.

Action Manager shutdown: The Action Manager is stopped using the `stopAM.bat(sh)` script located in the `TDI_install_dir/bin/amc` directory. This script uses the processID of the started AM to kill it. The processID is obtained by the `startAM` script and is stored in a file, which in turn is read by the `stopAM` script.

To stop the Derby database, type `stopNetworkServer`, which stops the Derby database server in Network mode. This should be done after AM is stopped, not before.

AMC Logs

The Administration and Monitoring Console logs are stored in the ISC log in the environment in which AMC runs:

- for ISC SE, the log file is created under `${LWI_HOME}/logs`;
- for Tivoli Integrated Portal (ISC embedded), the logs are logged in `${WAS_HOME}/profiles/ ${profileName}/logs/${serverInstance}/SystemOut.log`

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).

You can view and delete AssemblyLine logs in the **AssemblyLine Logs** window. To reach **AssemblyLine Logs** on the **Monitor Status** window: (**Monitor Status**→ **Solution View Details**→ **View Logs**). In the **Solution View Details** window, select the AssemblyLine whose logs you want to view. In the **AssemblyLine Logs** window, select any logs you want to delete, and select **Delete**. You can delete one or multiple logs. To view a log, click its hyperlink.

The **Solution View Details** window also contains the **Action Manager Logs** table. You can select and delete logs from **Action Manager Logs**.

You can manage all of your logs in the **Log Management** window. You can specify criteria for displaying logs, and you can delete logs for all AssemblyLines or for a single AssemblyLine. You can select to delete all logs for AssemblyLine(s) specifying a date range, or you can delete the most recent logs, where you enter the number of most recent logs.

Compatibility with previous versions of Tivoli Directory Integrator

When Administration and Monitoring Console for Tivoli Directory Integrator 7.1.1 is used to manage older versions of Tivoli Directory Integrator (TDI), some functionality is limited.

- The concept of Tivoli Directory Integrator Properties was changed in TDI 6.1. For this reason, AMC does not support editing/viewing/management of properties when working with a TDI 6.0 server. Also, the properties related triggers and actions are unavailable when working with a remote TDI 6.0 server.
- The **sendEventNotification** API for sending events remotely to a Tivoli Directory Integrator Server was introduced in Tivoli Directory Integrator 6.1. Therefore AMC does not support sending of Tivoli Directory Integrator events to remote TDI 6.0 servers. The Send Event Notification Action (in Action Manager) windows are not available when working with a Tivoli Directory Integrator 6.0 server.
- The Tombstone Manager Feature was introduced in Tivoli Directory Integrator 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 Tivoli Directory Integrator 7.1.1, by adding an API in Tombstone Manager. Hence, this feature is available only in Tivoli Directory Integrator 7.1.1.
- 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" window, only those Configs that are already loaded are shown on the remote TDI 6.0 server. A Load operation is not available for TDI 6.0 servers. Only Stop and Re-load operations are available for TDI version 6.0.
- The AL Operation feature was introduced in TDI 6.1. Therefore starting an AL with AL Operation is not supported for TDI 6.0 servers.
- You are able to Add a Solution View (minus properties), select ALs to expose, select health AL, configure users, start or stop ALs, stop or 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 window supports username and password fields for this purpose. If a user attempts to pass username and password to a TDI 6.0 server, AMC is not able to connect to the remote Tivoli Directory Integrator 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 Solution View:
 - Publish Solution: is enabled only for TDI server version 6.1.1 or later, and if there is actually a published solution defined for the selected config instance.
 - Create Solution View with All Assembly lines exposed: this feature is available with TDI 6.0, 6.1 and 6.1.1 servers.

- Creates a Solution View with all AssemblyLines from the config instance exposed, and all properties and no Health AL defined. This is similar to a quick start type of option. This option is disabled for TDI 6.0 servers (because TDI properties are not available in TDI 6.0 Servers)..
- In the View Tombstone window only the 30 most recent tombstones entries are displayed.

AMC in the Integrated Solutions Console

The Integrated Solutions Console (ISC) is designed to offer a common console to organize administrative console functions using industry-standard technologies. Starting with TDI 7.0, integration of the AMC into the Integrated Solutions Console (ISC) comes with the following changes. The primary navigation links for AMC are:

- **Administration and Monitoring Console**
 - Servers
 - Solution Views
 - Monitor Status
 - Action Manager
- **Advanced**
 - Log Management
 - Console Properties
 - Preferred Solution Views
 - Property Stores

Console user authority

Using ISC **Console User Authority**, you can add and remove users to AMC. In the AMC for Tivoli Directory Integrator v7.0 and later, the following are the roles:

Table 27. AMC roles

User Role	Description
administrator	Users with this role assigned are able to configure the roles other users are assigned to.
iscadmins	Users with this role assigned have the ability to control the settings of the ISC console itself.
TDI AMC Admin	This role is considered by the TDI AMC application deployed in the ISC console. Users with this role assigned are able to administer Servers, Solutions View roles, Manage Console Properties. (This was the superadmin role in the AMC prior to Tivoli Directory Integrator 7.0)
TDI AMC User	This role is considered by the TDI AMC application deployed in the ISC console. Users with this role assigned are able to use the provided by the TDI AMC Admin resources.

Within the *TDI AMC user* role, user privileges are assigned roles found in the Solution View:

- Admin
- Config Admin
- Execute
- Read

The roles control access to functions on the console. You can see only those functions for which you have roles assigned. For example, users with the TDI AMC Admin role automatically have administrator privileges over all Solution Views. Administrators can configure the properties required for the Web administration tool (modifying properties related to `amc.properties` file, available from Console Properties in the left navigation pane). A user with the TDI AMC User role in ISC is the same as the current non-admin AMC user. TDI AMC Users cannot access any administrative windows such as TDI Servers and Console Properties.

ISC features the AMC Admin Group or `iscadmins`. A user in the `iscadmins` group has the same privileges as the administrator.

Administrator and the `iscadmins` group

The administrator, defined as the user who has installed the application, can manage AMC users. The administrator can add or remove users from the local OS Registry to the AMC application and assign or edit roles. Tivoli Directory Integrator v7.0 and later offers a new **TDI AMC Admin** group. The superadmin role does not exist after TDI version 6.1.1.

Action Manager

The Action Manager 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 Action Manager window 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 28. Action Manager triggers

Trigger Type	Trigger Details	User Input for trigger
No trigger	A rule with this triggering type has no triggering condition, and as a result never gets triggered by itself. The only way this rule can be executed is if some other rule executes this rule	No details required
On AssemblyLine start	A rule with this triggering type gets triggered when the Action Manager receives an AL start event for this particular AL.	"AssemblyLine name"
On AssemblyLine termination	A rule with this triggering type gets triggered when the Action Manager receives an AL termination event for this particular AL.	"AssemblyLine name"
On config load	A rule with this triggering type is triggered when a Config is loaded.	"Name of Config to load"
On config unload	A rule with this triggering type is triggered when a Config is unloaded. The Config must be loaded to be unloaded.	"Name of Config to unload"

Table 28. Action Manager triggers (continued)

Trigger Type	Trigger Details	User Input for trigger
On query AssemblyLine result	<p>Note: A rule with this triggering type should not be used with a short-running AL. This is because Action Manager stores the handle of the AL object on receiving the Start AssemblyLine event. Later on, receiving the Stop AssemblyLine event, Action Manager uses this handle to query the final work entry attributes. If the AL terminates before Action Manager can store the handle, then Action Manager is not able to query the work attributes. Usually an execution time of 10 seconds is sufficient (this can be achieved by putting a system.sleep(10) before the AL terminates, for example in the epilog hook).</p> <p>When running On query AL result, Action Manager polls the AL continuously for the specified polling interval. The trigger first checks for the attribute value, starting the AL after the specified polling interval. Next, the trigger checks the AL result entry.</p> <p>On query AL result is a rule that is triggered when the last "work" entry of the specified AL contains the specified "Attribute" matching the given "condition" and "value". This condition is checked only when the ActionManager receives a Stop AssemblyLine event. The user can specify a time interval. The specified AL run periodically depending upon the time interval specified. 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 is checked only when the Action Manager receives a Stop AssemblyLine event.</p> <p>To configure the Query on AL result trigger, enter values for the following fields:</p> <ul style="list-style-type: none"> • AssemblyLine name • Attribute • Condition • Value • Polling Interval • Polling Unit 	'AssemblyLine name', 'Attribute', 'Condition', 'Value', 'Polling Interval,' and 'Polling Unit.'
On server API failure	A rule with this triggering type is triggered when the Action Manager is unable to connect to the remote server using the Server API. You can configure different polling time intervals for each Assembly Line depending upon AL execution time.	'Polling Interval' and 'Polling Unit.'
On received Event	<p>A rule with this triggering type is triggered when the Action Manager receives an event which satisfies the criteria mentioned.</p> <p>Note: If any of the criteria are to be ignored, just leave it blank.</p>	'Event type', 'Event Source', 'Event Data'. Event Data is optional. Event type or source – one of them must be specified.

Table 28. Action Manager triggers (continued)

Trigger Type	Trigger Details	User Input for trigger
On Property	<p>A rule with this triggering type is triggered when the specified property meets the specified condition. The Action Manager periodically checks for this property. You can configure a polling interval and polling units when configuring this trigger.</p> <p>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.</p>	"Polling Interval," "Polling Unit","Property name", "Condition", and "Value".
On local variable	<p>A rule with this triggering type is triggered when the specified variables meet the specified condition. The Action Manager periodically checks for this property .</p> <p>Note: This rule gets triggered only once, and gets reset back to the ready state only when Action Manager detects that this variables 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.</p>	"Local Variable" , " Condition", "Value".
Inspect AssemblyLine exit code	<p>A rule with this triggering type is triggered when an AssemblyLine terminates with an error.</p> <p>Inspect AL Exit Code also searches for an error object string for every abnormal AL termination. Under Configure Trigger, if the trigger is Inspect AL Exit Code, you can enable Inspect Error Object. In the Value field, type the string you want for Error Object. Note that if the Value field is empty, then the rule triggers for every abnormal termination of an AL. If Inspect Error Object is not selected, the trigger waits for the AL to terminate and inspects the exit code for an attribute value (entered by the user). Type values for both the Attribute Name and Value.</p> <p>In the Inspect AL Exit code trigger, the Action Manager no longer starts the AL, and there is no polling. The trigger only checks the AL result one time after the AL runs.</p>	"AssemblyLine name"; if "Inspect Error Object" is enabled, you only need supply "Value." If "Inspect Error Object" is disabled, values for "Attribute," "Condition," and "Value" are needed.
Time since last execution	<p>A rule with this triggering type get triggered when the Action Manager detects that the specified assembly line has not run for the specified period. Note: This rule is triggered only once. After that Action Manager wait for receiving a Start AssemblyLine 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.</p>	"AssemblyLine name", "Not Run Since" and "Unit".
Timer	<p>A rule with this triggering type is triggered continuously within an interval defined by a number of units and the measure of seconds, minutes, hours or days.</p>	"Interval" and "Unit."

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 29. Action Manager actions

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 Rule	This action Enable or Disable the chosen rule.	"RuleName" "State"
Execute Rule	This action cause the execution of the specified rule, which in-turn imply execution of all the actions specified in that particular rule.	"RuleName"
Notify Event	This action cause the Action Manager to emit an event with the specified details to the Server associated with the current Solution View. See the Session.sendCustomNotification() API for details.	"Event type", "Source", "Data".
Modify Property	This action cause the Action Manager to modify the selected property based on the specified operation.	"Property", "Operation", "Value".
Copy Property Value	This action 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 causes 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 -> Solution View Details -> Action Manager Results table. It is advised to always have at least one Log action (containing descriptive text) in every rule.	"Severity", "Message", "Description".
Send Email	This action causes an email to be sent to the recipient you specify. You supply the content of the email. Along with the content, the Action Manager provides other details before sending the mail. In the content input area as well as in the subject line, you can specify the variable %EVENT_DATA% value. Specifying %EventData% inserts the actual value of the Eventdata variable when the mail is sent. %Action_Error% can also similarly be substituted here. If Attach Action Manager Log is enabled, the Action Manager logs (as specified in the am_logging.properties file) are sent as an email attachment.	"To", "From", "Subject", "Attach Action Manager Log" (Selected/Not Selected), "Content".
Modify local variables	This action causes the action manager to increment, decrement or set the value of the specified variable to the specified value.	"Variable", "Operation", "Value".

Table 29. Action Manager actions (continued)

Action	Action Details	User Input for action
Execute command	This action causes the specified command to execute on the target computer. The command can be any generic command or a TDI specific command.	"Target Machine", "Port", "Username", "Password", "Keystore", "Keystore Password", "Protocol", and "Command".

Rules that are configured for Solution Views in AMC, are stored in AMC's Derby 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 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 thread checks for the server accessibility. If the server has become inaccessible, all rules threads created for the server are terminated (except for those with triggering type 'On Server API failure'). Similarly if the server becomes accessible, rule threads are 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 are 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, and so on.) rule threads are terminated, created or refreshed.
5. DatabaseModificationListener - This database listener thread continuously monitors addition, modification or deletion of rules. Whenever any changes in the rules are detected, the Action Manager 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 Log action is configured for the rule, then that also gets logged into the AMC database. These database entries are used to show the appropriate status in Monitor windows of AMC.

Enabling Action Manager

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 Derby 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:1528/C:/Program Files/IBM/AppSrv  
/profiles/amcprofile/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  
{protect}-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=1528
```

Note: Both AMC and AM support alternative databases, like MS SQL, Oracle and so forth. In order for AMC and AM to connect to one of those alternative databases, the configuration statements in `amc.properties` and `am_config.properties` will look very different.

When the Action Manager is started, it attempts to connect to AMC's database. If it fails in performing any initial setup tasks, it exits with an exception message. Check the `amc.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 Derby DB in network mode.

SSL settings and encryption properties for AM are configured in the following set of properties:

```
# Action Manager SSL properties  
javax.net.ssl.trustStore=TDI_Install_dir/serverapi/testadmin.jks  
{protect}-javax.net.ssl.trustStorePassword=administrator  
javax.net.ssl.trustStoreType=jks  
javax.net.ssl.keyStore=TDI_Install_dir/serverapi/testadmin.jks  
{protect}-javax.net.ssl.keyStorePassword=administrator  
javax.net.ssl.keyStoreType=jks  
# Action Manager encryption properties  
com.ibm.di.amc.am.encryption.keystore = TDI_Install_dir/testserver.jks  
com.ibm.di.amc.am.encryption.key.alias = server  
com.ibm.di.amc.am.encryption.keystoretype = jks  
com.ibm.di.amc.am.encryption.transformation = RSA  
com.ibm.di.amc.am.stash.file = TDI_Install_dir/idisrv.sth
```

These properties are similar to the encryption properties used by the server. For convenience the location of the stash file has been added as a property: `com.ibm.di.amc.am.stash.file`. By default the AM will reuse the server's keystore and stash file for encryption/decryption of AM protected properties.

Further configuration of run-time rules, triggers and actions is described under "Action Manager" on page 243.

Action Manager status in real time

When you log in to AMC, a one line Action Manager status displays in the **Welcome to AMC** panel. The Welcome to AMC panel displays the Action Manager status in a link, for example, "Action Manager is running" or "Action Manager is not running." To launch the **Action Manager Status** window, click the hyperlink. The Action Manager Status window displays Action Manager Status in real time, as well as thread details and trigger details. This window displays status information in real time. This window shows:

- Action Manager Status, for example the Boot Time
- Action Manager Thread Details
- Action Manager Trigger Details

The AMC directly queries the Action Manager using the APIs exposed by Action Manager. If AMC cannot establish a session with the Action Manager, the AMC concludes that the Action Manager is not available because the Action Manager has stopped. In addition to Action Manager status, AMC displays details of thread information and trigger details.

Action Manager creates a number of threads. Some Action Manager threads monitor the essential functionality of the Action Manager such as the Database Modification Listener and the ServerStatusListenerThread. Moreover, from these threads the Action Manager creates threads for each of the trigger rules that is configured in AMC. With the Remote Method Invocation (RMI) Layer, AMC can query the status of the various trigger-related threads. Using the RMI based query, AMC knows the state of these threads, thread priority, and so on. AMC can also query the triggers that have been executed over a period of time.

Two new properties belong to the **Display real time Action Manager Status** requirement. The properties that allow AMC to display the Action Manager status in real time are `am.api.host` and `am.api.port`. Action Manager status used an RMI layer around the Action Manager that exposes an API to be used by AMC for querying the Action Manager for its status.

AMC force trigger for a given rule

AMC allows users to force a trigger for a specific rule. Forcing a trigger gives the user an idea of what the Action Manager does when the rule is triggered. When you select **Disable Rule**, the selected rule is disabled. When you select **Force Trigger**, actions configured for the selected rule are executed.

Action Manager can execute a set of actions configured for a particular trigger (rule) explicitly. The AMC user does not have to wait for the triggering condition of a rule to be satisfied before the configured actions are carried out. Users can define actions that are to be executed so that they can test those actions. Users can execute all of the supported actions using **Force Trigger**. However, the **Revert** action is effective for only some (a subset of) the supported actions.

- Modify Property
- Copy Property
- Write to Log
- Enable Disable Rule

AMC and Action Manager security

Introduction

The Administration and Monitoring Console (AMC) is a web-based application for monitoring and managing remote Tivoli Directory Integrator solutions. The following features of AMC have been improved:

- Encryption or concealment of passwords that are stored in the `amc.properties` file
- Use of stash files to store keystore passwords
- Enablement of the `BUILTIN` authentication scheme in the Derby database

AMC uses the Remote Server API to communicate with Tivoli Directory Integrator. For this reason, all the security restrictions and configuration settings that are applicable to Tivoli Directory Integrator Remote Server API clients (as mentioned in previous sections) are valid for AMC too.

Action Manager is installed with AMC. Action Manager configures itself and behaves based on rules set in the AMC database by AMC users. To monitor remote AssemblyLines and to take action based on configured rules, Action Manager, just like AMC, uses the Tivoli Directory Integrator Remote Server API to communicate with Tivoli Directory Integrator servers.

Note: Communication between AMC and AM using RMI is not protected in any way.

AMC and SSL

Multiple Tivoli Directory Integrator servers can be registered with AMC. Each TDI server may be configured differently; one Tivoli Directory Integrator 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 Tivoli Directory Integrator to run in SSL mode the `api.remote.ssl.on` property should be set to `true` in `global.properties` (or `solution.properties`).

As AMC is a web application running inside a Web Container it automatically inherits some properties and security restrictions from the Web Container. For instance, if the Web Container has an SSL keystore or SSL truststore configured, then that would be automatically applicable to AMC. But AMC can also override that – and specify its own keystore and truststore.

For being able to communicate with Tivoli Directory Integrator Remote Server API running on SSL, AMC must have a keystore configured which contains the certificate that is trusted by the Tivoli Directory Integrator remote Server API (that is, it must be present in Tivoli Directory Integrator's truststore's trusted certificates section) and AMC must have a truststore configured which contains the certificate that is sent by the Tivoli Directory Integrator remote Server API. In other words – the certificate that is present in Tivoli Directory Integrator server's keystore must be present in AMC's truststore and the certificate that is present in Tivoli Directory Integrator truststore must be present in AMC's keystore.

For example, the default installation of Tivoli Directory Integrator is shipped with certain stores (.jks files). When you run Tivoli Directory Integrator in SSL mode, then to connect to AMC its keystore and truststore must both be set to the same value: `TDI_install_dir/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 that you set up your own SSL keystores and truststores.

In AMC, the path of the truststore and keystore can be set by logging into AMC as "TDI AMC Admin" (Console Administrator) and navigating to the following window: **Advanced -> Console Properties -> SSL settings**. The settings for truststore and keystore are written to `amc.properties` file inside the `tdiamc` folder in Web Container. You can alternatively choose to edit the `amc.properties` file directly. With Tivoli Directory Integrator 7.0, AMC can be deployed in ISC Standard Edition (SE) or in ISC Advanced Edition (AE). Depending on the ISC runtime, the location of the `testadmin.jks` file varies. For example, if AMC is deployed in ISC SE, then the location will be `ISC_RUNTIME_INSTALL_DIR/runtime/isc/eclipse/plugins/AMC_7.0.0`. On the other hand, if AMC is deployed in Tivoli Integrated Portal (ISC embedded), then the location is `ISC_RUNTIME_INSTALL_DIR/systemApps/isclite.ear/tdiamc.war`. The keystore and truststore password are set to "administrator" by default. To establish an SSL based connection with a remote Tivoli Directory Integrator server, you must start the server in "SSL enabled" mode, and for a Non SSL based connection, start the server in "SSL disabled" mode.

Attention: Default SSL settings are provided. However, using the default certificates does not increase the security more than just using a plain connection, so after installation, you should replace the default SSL certificates and update the keystores and truststores accordingly in order to increase security.

For each Tivoli Directory Integrator server running over SSL that you wish to register with AMC, you must import the necessary certificate into AMC's truststore and the necessary AMC's key certificate into Tivoli Directory Integrator's truststore. 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 is `http://hostname:port/ibm/console`.

Action Manager monitors running Configs and AssemblyLines on remote TDI Servers based on rules configured in AMC. Action Manager ships with the keystore and truststore 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 Tivoli Directory Integrator Servers remotely. Each Server can be configured in one of the following ways:

- Non SSL
- SSL
- Custom Authentication with Non-SSL
- Custom Authentication with SSL

This section looks at each of these cases in detail.

When a remote Tivoli Directory Integrator server is configured for non SSL (that is, `api.remote.ssl.on=false`) then the keystore or truststores 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 computer 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 computer's IP address is entered in the `global.properties` file of the remote TDI server, AMC is able to connect to that particular server. It is a way of saying – I trust remote server connections (AMC connections) from only those computers whose IP addresses I have mentioned in my `api.remote.nonssl.hosts` property.

Note: If the Tivoli Directory Integrator server is running on the same computer as AMC, then editing this property is not required.

When a remote Tivoli Directory Integrator server is configured for SSL (that is, `api.remote.ssl.on=true`), then the SSL keystore and truststore 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 must be passed while making a connection to the remote TDI server. The remote TDI server validates this user name and password against some third party repository like LDAP, file, database, script, and so on 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 (**Servers -> Modify Server**) in the Authentication mode window – select **LDAP or Custom Authentication** and enter the Username and Password that AMC must pass every time it attempts to connect to the specified remote TDI Server.

Note: If the Username or Password (in case of custom authentication) or SSL keystores or truststores (in case of SSL) are not set up correctly, then AMC is unable to connect to the remote TDI Server and show that server as "Stopped" or "Not running."

AMC and role management

Every user (or group) in AMC can be assigned one of the following roles in AMC for a particular Solution View. This role assignment can be done in the **Solution Views** window by selecting a particular Solution View and by clicking **Configure Access Control Lists (ACLs)**. The **Configure ACLs** window displays. Select the Name of the user you want to configure and click **Configure Users** on the toolbar. The **Configure Users** window displays. Select the **User ID** and select one of the available roles:

- Read
- Execute
- Admin
- Config Admin

Note: You must reload Solution Views created using the Auto Update option. Use the **Refresh Solution View** in the **Solution Views** window. For Solution Views marked for auto update, you must reload the config file and refresh the Solution View by clicking the **Refresh Solution View**. If a user fails to refresh a Solution View created using the **Simple** option and flagged for auto update, the Solution View may cause inconsistencies in the AMC database. Inconsistencies in Solution Views that are not updated could result in incorrect behavior by the Action Manager.

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 Solution View, definitely is available to a user with "Execute" privilege on that Solution View. Any functionality that is available to a user with "Execute" privilege on a Solution View, is 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 Solution View – such as what are the ALs inside this view, what are properties inside this view, what is the status of these ALs, and so on. This user cannot modify, start, stop, or change any detail of this Solution View.

Execute

This is essentially a Read user with one extra privilege – the ability to Start and Stop AssemblyLines.

Admin

This user can administer the Solution View, without being able to modify the Solution View itself. This user can do everything that the "Execute" privilege user can do, and additionally he can modify properties, delete logs, configure rules, and so on, for this Solution View.

Config Admin

This user can virtually do anything to the Solution View – including modifying the view itself, modifying the permissions of other users on this view, and so on. This is the highest privilege that can be given to a user for a particular Solution 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 Solution View "SynchDatabase", then both "test" and "tdi" automatically get ConfigAdmin privilege over the "SynchDatabase" Solution View.

Notes:

1. If the "test" user is explicitly given "read" privilege for the same Solution View, then "Read" gets 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 Solution View – then in this case the test user gets the higher of the two privileges – in this case being "admin", unless a specific role is already assigned to "test" for the same Solution View – in which case the specific role assigned to "test" is given precedence [point 1 above].

AMC and passwords

Any password field that is stored in the amc.properties file, such as LDAP Bind password, keystore password, and so on, are all encrypted before being written to the amc.properties file. Also, AMC never displays any Password fields or protected fields on console. All such fields are masked out.

AMC and encrypted configs

AMC allows users to load and connect to password protected configs. On the Load Reload window of AMC, a password text box has been provided – where the users must enter the password of the config

they are attempting to start before clicking **Start**. Similarly, in the Action Manager Screen – for the Start AssemblyLine action, a password field has been provided where the user can enter the password of the config. Action Manager passes 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 just see an error message saying – "Unable to start the config". The user can see the TDI Server logs where an exact message is provided.

Administration and Monitoring Console User Interface

Log in and logout of the console

Open a Web browser and type the following address:

`http://hostname:port/ibm/console`

Where *port* is the port where your Web server is running. When deployed on the bundled web container the ports by default are 13100 for HTTP and 13101 for HTTPS communication.

The login page can also be launched by using the `launchAMC.html` file which is placed in the `TDI_install_dir/bin/amc` folder. This file is not present on i5/OS.

The IBM Tivoli Directory Integrator Administration and Monitoring Console login page window is displayed.

Logging on to the console as the console administrator:

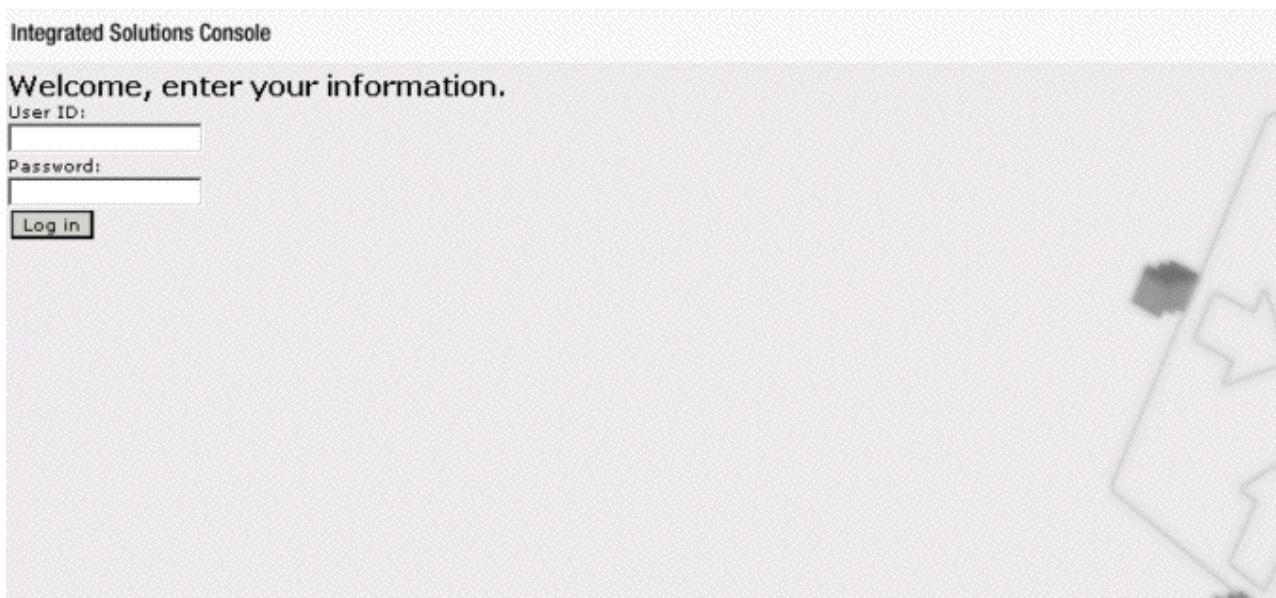
The console administrator is a user who can:

- Configure the properties required for the AMC
- Set the authentication mechanism used for AMC logins
- Add new users and configure users' roles

When logging in for the first time use the system username and password you have installed Tivoli Directory Integrator, AMC and the embedded web platform with. If you have deployed AMC in WAS/Tivoli Integrated Portal (ISC embedded) then you would need to log in with a user that has been assigned the *iscadmins* and *administrator* roles.

Note: The embedded web platform uses the PAM authentication mechanism on UNIX and Linux boxes to validate the system username and password provided on log in. This is why on AIX machines you must have the `auth_type` parameter set to `PAM_AUTH` in the `/etc/security/login.cfg` file.

To log in to the Integrated Solutions Console, type your user name and password in the boxes provided in the login window and click the **Log in** button.



The **Logout** button is in the upper right hand corner of the console, next to **Help**. When you click **Logout**, you are returned to the Log in page.

AMC Console Layout

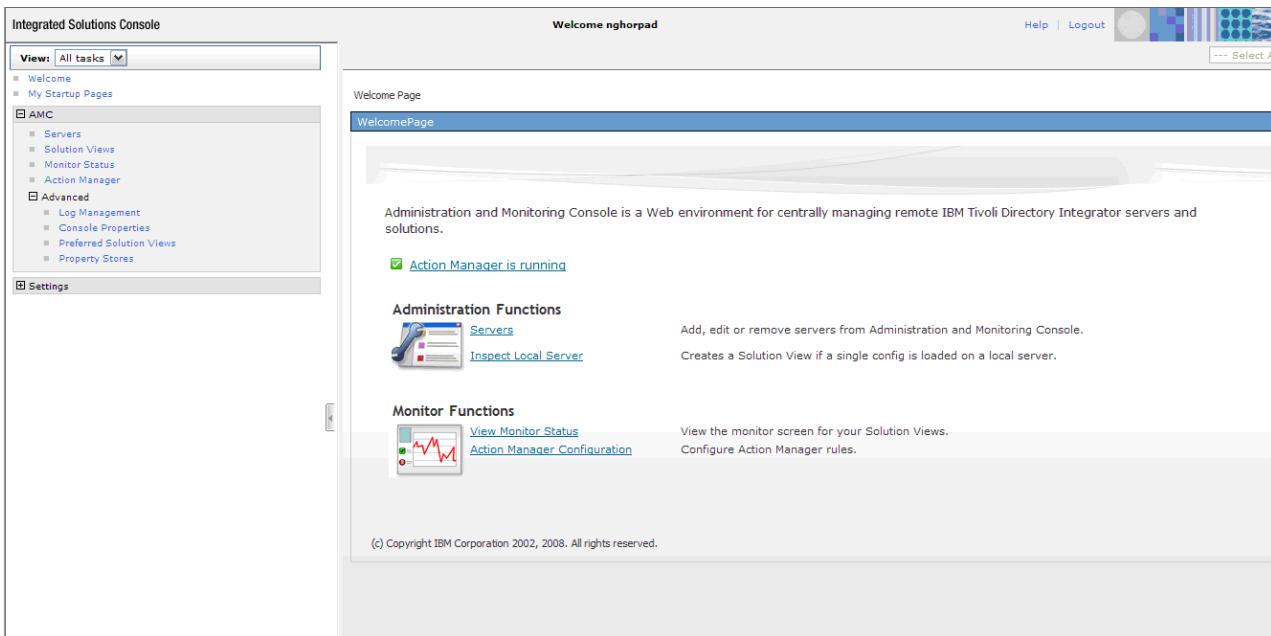
The IBM Tivoli Directory Integrator Administration and Monitoring Console includes the following components:

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.



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.

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” on page 233 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 233 for more information.
- Click the **Clear all filters** icon clear all filters set for the table. See “Filtering” on page 233 for more information.
- Click the **Edit sort** icon to sort the information in the table. See “Sorting” on page 232 for more information.
- Click the **Clear all sorts** icon to clear all sorts set for the table. See “Sorting” on page 232 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.
- Click the **Export** icon to export the table data.

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** window, 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 **Match case** 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 **Match case**.
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.

Servers

This window 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 window, as well as launch the **Config Files** window.

When AMC is started , it automatically has a local Tivoli Directory Integrator server, registered on port 1099 . Therefore, in the **Servers** window, one entry in under LOCAL SERVER is already present with its state depicted as running or unavailable depending on its status.

To load or reload a config, select **Servers** and click **Config Files** in the toolbar of the Servers window. The **Config Files** window appears.

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 Add on the toolbar.

Delete Select the radio button next to the server you want to delete and click **Delete** on the toolbar.

Modify

Select the server for which you want to modify information and click **Modify** on the toolbar.

Config Files

Select the server for which you want to list configuration files. When you click the **View Config Files** link in the Solution Views window, it launches the Config Files window. Each configuration file is labelled as loaded or not loaded. The toolbar provides a variety of load, unload, and reload options.

Shutdown server

Select the server you want to shut down and click the Shutdown Server on the toolbar.

Shutdown gracefully

Shuts down a running server gracefully (create new Threads that wait for the AssemblyLines to stop).

Note: Graceful shutdown is not supported for Tivoli Directory Integrator servers earlier than v7.1.

Add a server

This window 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 use features on other AMC windows to add Solution Views to the TDI server and to create and define views for the Solution 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 LDAP or Custom authentication method, enter the username and password to be used for authentication.
5. Click **OK**.

Modify a server

This window allows you to edit the information for an existing IBM Tivoli Directory Integrator server. To edit an existing server:

1. Look at the displayed Server ID. If you want to change the Server ID, click **Change Server ID**.
2. Type the Name for the server.
3. Enter the host name or IP address of the computer on which the IBM Tivoli Directory Integrator server is running in the **Hostname** field.
4. Enter the port number on which the IBM Tivoli Directory Integrator server is configured to run.
5. Select the desired authentication mode. If you selected the LDAP or Custom authentication method, enter the username and password to be used for authentication.
6. Click **Cancel** to exit the window without making any changes, or click **OK** to save the changes.
7. Click **Test Connection** to see whether the connection to the server succeeds or not based on the current settings.

Console Properties

Use the Console Properties window of AMC to manage configuration information such as database configuration of AMC, SSL settings, Action Manager log rotation frequency, and so on.

General

Use the General window of AMC to set log rotation frequency of Action Manager in days.

SSL

Use the SSL window to configure SSL settings for AMC. The SSL settings apply to AMC's SSL connection to the remote Tivoli Directory Integrator server. The SSL properties that are exposed are only the AMC's keystore and the trust store properties. If SSL is turned on in the remote server, an administrator needs to make sure that the required certificate is imported in his store for the connection to work. An administrator should import each remote server's certificate that he wishes to connect to in his store.

JDBC Properties

JDBC properties are used to define the connections settings to the Derby database, or to other databases compatible with the Administration and Monitoring Console, such as Oracle and MS-SQL Server. The AMC database stores AMC configuration information, connection details, and Action Manager rules and results.

The IBM Tivoli Directory Integrator AMC supports alternative databases in addition to Derby. AMC bundles the Derby database. AMC communicates with its database using the Java Database Connectivity (JDBC) protocol. JDBC is a generic protocol and can be easily extended to other databases. AMC support for alternate databases enables you to have AMC installed and communicating to an existing database. The database stores Action Manager logs, results, and so forth. The Integrated Solution Console **Advanced -> Console Properties** section groups the **JDBC properties** to Derby or to another database. In the case of Derby, you can configure the database to run in both embedded as well as network mode. The default database is Derby and the default mode is network mode.

From this window you can:

- Select a database from the **Database Type** field, options are Derby, MS SQL Server, Oracle and DB2.
- Type the a value for the JDBC URL in the **JDBC URL** field.
- Type the user name for the database in the **Username** field.
- Type the password for the database in the **Password** field.
- Type the JDBC driver name in the **JDBC Driver** field.

As for the JDBC URL and JDBC Driver parameters, the following table provides some guidance:

Table 30. Driver parameters

Database	JDBC URL	JDBC Driver	Driver .jar file
Derby	jdbc:derby://host:port/database [;create=true create=false]	org.apache.derby.jdbc.ClientDriver	derby.jar
MS SQL Server (2005)	jdbc:sqlserver://host:port; databasename=database	com.microsoft.sqlserver.jdbc. SQLServerDriver	sqljdbc.jar
Oracle	jdbc:oracle:thin:@host:port:database	oracle.jdbc.driver.OracleDriver	ojdbc14.jar
DB2	jdbc:db2://host:port/database	com.ibm.db2.jcc.DB2Driver	db2jcc.jar

Notes:

1. Depending on the database selected the corresponding driver .jar file must be copied to *TDI_install_dir/lwi /libs*.
2. Configuration of the Action Manager is also needed in order to specify the new database from where it will work. The same .jar file must be added to *TDI_install_dir/bin/amc/ActionManager/jars* and adjustments must be made to the *am_config.properties* file.
3. If you decide not to use Derby, but one of the alternatives, keep in mind that the database specified in the JDBC URL must already exist before you start AMC (otherwise AMC won't be able to create one and populate it). This is not needed if Derby is used because it supports the "create=true" option in the JDBC URL, thus causing AMC to automatically create the database (if it does not exist) when started.

Solution Views

Use the Solution Views window to view, Add, Modify, and Delete Solution Views.

- To add a Solution View, click the **Add** button on the toolbar.

- To modify an existing Solution View, select the Solution View and click **Modify**. Follow the steps in the **Modify Wizard**. Under **Modify Solution View**, click **Next** to go to the next step, and click **Finish** when you have completed the steps.
- To configure Access Control Lists for a Solution View, select the Solution View for which you want to configure ACLs and select **Configure ACLs...** on the toolbar.
- To delete an existing Solution View, select the Solution View you want to delete and click the **Delete** button on the toolbar.
- To launch a separate panel to Add / Edit / Modify local AM variables for that Solution View, click **Local Variables....**

Note: You must reload Solution Views created using the Auto Update option.

When you **Modify a Solution View** AMC checks to see if the Solution View was created using **Auto Update**. If the Solution View selected for modification was created using Auto Update, a message appears, saying:

The selected Solution View is marked for auto update. Ensure that auto update is disabled to modify the Solution View.

Solution Views are listed in the Solution Views table. If a specific Solution View was created using Auto Update, a >> short menu appears when you click on the arrows up and to the right of the Solution View name. You can select **Refresh Solution View** or **Disable Auto Update**. For Solution Views marked for auto update, you must reload the config file and refresh the Solution View by clicking the **Refresh Solution View**. If a user fails to refresh a Solution View created using the **Simple** option and flagged for auto update, the Solution View may cause inconsistencies in the AMC database. Inconsistencies in Solution Views that are not updated could result in incorrect behavior by the Action Manager.

Configure ACLs

From this window you can set the Access Control Lists (ACLs) for a user and associate that user with a specific Solution View.

- To configure a user or users, select the user or users you want to configure and click **Configure Users** on the toolbar.
 1. Select the user you want to 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 Solution View details like ALs, Tombstones, logs, properties belonging to the Config, and so on.
 - **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 Solution View, and assign and modify ACLs for other users.
 3. Click **Apply**.
- To remove an existing user, select the user from the table and click **Remove**.

When you are finished making changes, click **Apply**.

Local variables

Select Solution Views from the AMC left hand navigation pane. The **Solution Views** window appears. Select **Local Variables** from the toolbar. In the **Local Variables** window, you can select and **Add**, **Modify**, or **Delete** local variables for a Solution View.

The Action Manager triggers and actions must provide support for local variables that you can set or increment using rules and actions. Local variables can be used as triggering conditions for other rules. For example, a local variable can be set to a value of 1 and then can be incremented for every occurrence of the event and the local variable (in this example, the number 1 set to increment for every occurrence of

the event) – the local variable can trigger the rule "Terminate AssemblyLine". When the variable reaches a value of 10, you can configure a new rule to be triggered. The new rule could start a new AssemblyLine on a different server. Set these "local", AM-specific variables to a "Solution View". That means that the one variable created in a rule belonging to one Solution View can only be used in that Solution View's rules and is not accessible to rules of another Solution View.

Add a Solution View

The purpose of a Solution 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 Solution View to filter a configuration file for specific information so that only certain information within the configuration file is displayed. You can create multiple Solution Views for each Config, with each view exposing different information contained in the configuration file.

To Add a Solution View, select **Solution View** and select **Add** on the toolbar of the **Solution Views** window.

1. Enter view details:
 - a. Enter a name for the Solution View in the **Solution View Name** field.
 - b. Enter a description of the Solution View in the **Description** field.
2. Select the **Server** and **Configs** (configuration file) you want to use to create a Solution View:
 - From the **Server** menu, select the IBM Tivoli Directory Integrator server containing the configuration file you want to use to create a Solution View. This menu is 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 Solution View from the **Configs** list. The menu contains all currently loaded Configs.

Note: Click the **View config files** button to go to the Config files window. You can perform load or unload operations for the configs in this window.

3. Click **Add** on the **Solution Views** toolbar.
 - a. Type the name of the solution view you want to create in the **Solution View Name** field.
 - b. Type an optional **Description** for the Solution View you are creating.
 - c. Select the **Server** that contains the configuration file and AssemblyLines you want to use for creating a Solution View.
 - d. Select the configuration file you want to use from the **Configs** list.
 - e. Enable or disable **Auto Update**.

When the AssemblyLines or properties for a configuration change, **Auto Update** automatically changes the Solution View.

Note: When Auto Update is selected, you cannot edit the Solution View you created with Auto Update on, nor can you create Rules and Triggers for Solution Views made while Auto Update is on. If you want to edit the Solution View or add Rules and Triggers, you must disable Auto Update. The users would have to disable the auto update functionality in order to be able to create a Rules and triggers for Solution Views marked for auto updation. Review any changes to the config in Solution View by using the Refresh button on the Solution View window. This button is only visible to configs with auto-update set to true. Any config created manually using the Create Solution View wizard has the auto-update flag set to false.

Note: You must reload Solution Views created using the Auto Update option. Use the **Refresh Solution View** in the **Solution Views** window. For Solution Views marked for auto update, you must reload the config file and refresh the Solution View by clicking the **Refresh Solution View**. If a user fails to refresh a Solution View created using the **Simple** option

and flagged for auto update, the Solution View may cause inconsistencies in the AMC database. Inconsistencies in Solution Views that are not updated could result in incorrect behavior by the Action Manager.

4. Use the following options in creating a solution view:

Simple

Create a Solution View with common default options.

Auto Update

For Solution Views marked for auto update, you must reload the config file and refresh the Solution View.

Create Solution View from published solution.

Creates the Solution 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 7.0 server.

Create Solution View with all AssemblyLines exposed.

Creates a Solution 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 Solution View with all AssemblyLines exposed and all properties exposed.

Creates a Solution 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).

Create Solution View with all AssemblyLines exposed and all User properties exposed.

Creates a Solution View with all AssemblyLines from the config instance exposed, and all properties and no Health AL defined. This is similar to a quick start type of option. This option is disabled for TDI 6.0 servers (because TDI properties are not available in TDI 6.0 Servers)

Note: In order to be able to see user defined properties in the Property Stores panel you should do either of these:

- Place the .properties file in the folder containing the configuration file
- Specify an absolute path to the properties file when creating the property store in the CE (**New Property Store > Connector tab > Configuration tab > Collection path/URL parameter**)

5. Click **OK** to finish creating the Solution View.

Config files (allows loading/reloading of configurations)

To reach the Config Files window, and to access options for loading, reloading, unloading, and refreshing of config files, select **Solution Views** in the left navigation area. Select a server and a config file, then click the **View Config Files** button. This launches the Config Files window. This window displays loaded Configs and the Configs in the configs folder of the remote IBM Tivoli Directory Integrator server. When AMC is connected to a Tivoli Directory Integrator server, the Config Files window shows a listing of all files in the remote config folder (whether the files are valid TDI config files or not). You should perform Load operations on valid Tivoli Directory Integrator Config files only, otherwise an error message displays in AMC. The status Loaded or Unloaded displays with green (Loaded) and red (Unloaded) icons in the Status column. You can select one or more configs from the **Select** column of the Config files table. Once you have selected a config, you can **Load**, **Load As...**, **Unload**, **Reload**, or **Refresh** using the buttons at the top of the table. If you want to load a password protected Config, select the Config and type the password in the Password field.

Whether an action is successful or unsuccessful, a message displays after the action (Load, Load with Run name, Reload, Unload, and Refresh) executes, describing the outcome. For Load, Reload, and Unload, the new status for the configs that you selected displays in the Status column.

Note: You must have superadmin or config admin privileges to perform these actions.

- To load Configs, select the configurations you want to load and click **Load**.
- To load multiple instances of one Config, select the config you want to load and click **Load As....** The **Custom Load** window opens, allowing you to specify **Config File**, **Config Run name**, **Config Password**, and **Property Store Value**.
- To unload Configs, select the configurations 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 starts upon loading.

- To reload Config, select the loaded configurations you want to reload and click **Reload**. You can only reload a configuration that has the status of Loaded.
- To refresh Configs, click **Refresh**. Information for all of the configs in the table is redisplayed.
- Click **Close** when you are finished making changes.

Custom load: The Tivoli Directory Integrator server supports loading multiple instances of the same config with different run names. If you load config instances using **Load As....**, you can use these configs to create Solution Views and Rules. Use AMC to load multiple config instances by performing these steps:

1. From the **Welcome** page, select **Servers -> Config Files**.
2. Click **Load As....**
The **Custom Load** window appears.
 - a. Select the **Config File** from which you want to create multiple instances and click **Go**.
 - b. Type the **Config Run name**.
 - c. Type the **Config Password**.
 - d. Type the **Property Store Value** for each Property Store Name.
3. Click **OK** to use the values you have entered to create an instance of the config with the Run name you have specified. After an instance of the config is created, you are returned to the **Load Reload** window.
4. Click **Cancel** if you do not want to create the config with the values you have specified in the **Custom Load** window.

Note: Users must maintain data integrity.

- For example, if a Solution View and rules have been created for a config named config1.xml, and with a run name of ABC, do not load a different config, for example, config2.xml, with the name ABC either as a solution name or a run name.
- If you want to reuse Solution Views that you created using a specific run name and set of property files, you must unload this config using the same run name and property files.

Monitor Status and Action Manager

If you have not done so already, expand the **Monitor Status** category in the main navigation area of the Administration and Monitoring Console.

Do one of the following:

- To view information about each Solution View, see the Monitor Status table. Information regarding the Solution Views, such as Action Manager Status, Health Check Result and Health Check Status, display. You can also display **Solution View Details**, **Server Information**, and **Show Preferred Views**.
- To add, edit or delete Action Manager rules, click “**Action Manager**” on page 243.

Monitor Status

This window displays the views selected on the Preferred Views window accessed from **Advanced -> Preferred Solution Views**. It displays high level information about each preferred Solution View, such as:

Action Manager Status

Displays the status of the Action Manager rules for the selected Solution 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 Solution 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 Solution 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 is also 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 window you can:

- View Solution View details - To view the details of a specific Solution View, select the desired Solution View and click **Solution View Details**
- View Tivoli Directory Integrator Server Information - To view the details of the server to which the Solution View belongs, click **Server Information**.
- Show Preferred Solution Views - Click **Show Preferred Views** to view preferred Solution Views. This button is visible only if Preferred Solution Views are defined. You can define preferred Solution Views on the "Preferred Solution Views" window under **User Preferences**.

Solution View Details: The Solution View details panel in turn provides deeper view of the details specific to a Solution View which an administrator can take a look at and take action upon.

This window contains two tables. The top table displays the AssemblyLines associated with the selected Solution View and the status of each Solution View. The bottom table displays log information about recently triggered Action Manager rules.

When you are through making changes, click **Close**.

Solution View Details Table:

Columns: The **Solution 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 Tivoli Directory Integrator 7.1.1 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 Tivoli Directory Integrator 7.1.1 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 Tombstones - Select the AssemblyLine you want to view and click the **View Tombstones** button
- View Logs - Select the AssemblyLine you want to view and do one of the following:
 - Click the **View Logs** button on the toolbar.
 - Select **View 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 AssemblyLine -
 1. Select the AssemblyLine you want to start
 2. Click the **View pop-up** button
 3. Click **Start AssemblyLine**.
- Stop AssemblyLine - 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 AssemblyLine**.

Note: From Tivoli Directory Integrator v7.1 a new option is available – "Stop AssemblyLine gracefully". When selected the AssemblyLine will be stopped in a new Thread. Stopping AssemblyLine gracefully is not available for Tivoli Directory Integrator servers earlier than v7.1.

- Solution View Details - Click the **Solution View Details** button. Select the component you would like to view, for example, AssemblyLines.

Start AssemblyLine: Run the selected AssemblyLine.

Start AssemblyLine synchronously: AMC waits for the AL to terminate and shows the status of the run AL periodically. The output schema attributes of the AssemblyLine after its termination are viewable for synchronous AL runs.

Start AssemblyLine in simulate mode: The Assembly Line executes all components except for the connectors in the add, update, and delete modes. In essence, putEntry, modEntry and deleteEntry methods of connectors are not invoked in simulate mode. As a result, an Assembly Line running in simulate mode does not perform any additions, modifications, or deletions on third party repositories. For more information on simulate mode, see the corresponding section in *IBM Tivoli Directory Integrator V7.1.1 Users Guide*.

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 window displays useful information about tombstone entries, such as when the entry was changed to the tombstone state.

Delete Tombstones: On the **Monitor Status** window, select an AssemblyLine. Select the arrow to the right of the AssemblyLine and select **Delete Tombstones** from the menu. This launches the **Delete Tombstones**

window. The component details section of this window identifies the Solution View and AssemblyLine that are being worked on. In the choose delete criteria section, select one of the options to specify which tombstones you want to delete:

- Select **All Tombstones** to delete all of the tombstones for the selected AssemblyLine.
- Use **Start Date** and **End Date** to specify the date range from which the tombstones are to be deleted. AMC calculates the number of days from the selected date to the current date. AMC then deletes the tombstones generated for the calculated number of days.
- Use **Number of entries to return** to indicate a whole number indicating the number of recent tombstones to delete. When you click **Delete**, a confirmation message appears. When you confirm, AMC executes the delete command.

View Logs: Logs for a given AssemblyLine are displayed on the View Logs window. **Monitor Status -> Solution View Details -> View 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 Logs**.

Note: In order to view an AssemblyLine log in the Administration and Monitoring Console, the AssemblyLine must log using the SystemLog logger.

Action Manager 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 **Action Manager Results** table.

The following sections contain information about the **Action Manager Results** table columns and how to perform operations on Action Manager Results.

Columns: The **Action Manager 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 Action Manager 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**.

Server Information: This window displays the IBM Tivoli Directory Integrator server information of the server to which the currently selected Solution View belongs. The information on this window is read-only, although administrators have the capability to shut the server down from this window.

View Components: The View Components operation allows you to view the different connectors, function components and so forth configured in the selected AssemblyLine. N

Note: Branching components (IF, SWITCH, etc.) and Script components are not displayed. This is intentional design – attention is focused on Connectors/Function Components which are the key items.

Show Preferred Solution Views: Preferred Solution Views are the default Solution Views that are displayed on **Monitor Status** window.

Refreshing Solution View Details in AMC

The Solution View Details window is refreshed after a set interval of time to view the current AssemblyLine status. By default, the refresh rate is set to 600 seconds. The Integrated Solutions Console administrator has the privilege to change the refresh interval.

To change the refresh interval:

1. Go to the login page of AMC.
2. Type your user name and password, and click **Log in**. The Welcome page of Integrated Solutions Console appears.
3. In the left navigation tree, click **Settings** → **Manage Global Refresh**.
4. In the Manage Global Refresh window, click the **Monitor Status** link.
5. Change the refresh configuration settings and click **OK**.

Action Manager

This window allows you to add, delete or modify 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 window you can create an “Action Manager” on page 219 (or modify an existing one) for the current Solution 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.
- The set of alternate actions to be performed when the trigger is encountered.

Configuration rules settings: This window is concerned with the first part of the rule: defining triggers. From the window 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 AssemblyLine termination

Rule is triggered when the specified AssemblyLine is terminated.

On Config Load

Rule is triggered when the Action Manager receives a Config load event for this particular config.

On Config Unload

Rule is triggered when the Action Manager receives a Config Unload event for this particular config.

On Query AssemblyLine 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 Trigger

Rule is triggered when the specified property meets the determined Property name, Condition and Value specifications.

On Local Variable

Rule is triggered when the specified variables meet 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 variables does not meet the specified criteria any longer. The rechecking ensures that the rule is not repeatedly triggered for a single occurrence of the triggering condition.

Inspect AssemblyLine Exit Code

Rule is triggered when an AssemblyLine terminates abnormally. You can define an error object that Action Manager searches for in the AssemblyLine Exit Code.

Time since last execution

Rule is triggered when the specified AssemblyLine has not run for the determined period of time.

Timer Trigger

Rule is triggered continuously within the given interval.

Configure trigger: Each trigger type has a different selection of settings. If you do not see some of the fields listed below on your window, 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 modify actions. You can also move actions up and down in the table. For every action in the configured actions table that you can select, there is a column where you can enable the special trigger **Execute on Error**. **Execute on error** performs the action you have selected when an error condition occurs.

- To select an action to manage, enable the radio button that precedes each action that is listed.
- To add an action, click **Add**.
- To delete an action, select the action you want to delete and click **Delete**
- To modify an action, select the action you want to modify and click **Modify** .
- 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**.

Selecting **Execute on Error** carries out actions only if an error has occurred during the execution of any of the previous actions. You can use such actions to take corrective measures for handling any error that might have occurred during the execution of any previous actions. Action Error variables: AMC and Action Manager allow you to make the action error available in the various actions. At any point of time, if an error occurs during the execution of any configured actions, this error becomes available to you in the form of special reserved variables. You can then use these reserved variables in other actions you have configured. When the following actions are executed, Action Manager replaces the string %Action_Error% by the actual error that occurred during the execution of the previous actions. If no error occurs, the variable %Action_Error% is not replaced and stays as it is.

- Send Email
- Execute command
- Send event action
- Write Log action

Add/Modify Action: When a rule is triggered, the Action Manager executes the actions associated with the rule. This window 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 computer Action Manager is executing.

Select from remote config folder

Check box; 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 are 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 AssemblyLine 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 are prompted for the AssemblyLine's Initialization attributes and Operation attributes for this Operation. This label is shown only for TDI 6.1.X and Tivoli Directory Integrator 7.1.1 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 computer Action Manager is executing.

Select from remote config folder

; 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 are 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 Rule

Select the Enable/Disable Rule to enable or disable an Action Manager rule.

Rule name

Select the name of the rule-Solution View pair that you want the action "Enable/Disable Rule" to execute. In versions before TDI 7.1.1, you selected the rule name instead of a rule-Solution View pair, which is a new feature for 7.1.1. This option belongs to the action "Enable/Disable Rule."

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 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-Solution View pair that you want the action "Execute Rule" to execute. In versions before TDI 7.1.1, you selected the rule name instead of a rule-Solution View pair, which is a new feature for 7.1.1. This option belongs to the action "Execute Rule."

Execute Command

The Execute Command action can execute the command entered in the Command field on the target computer specified under **Target Computer Name**. The command can be any generic command or a IBM Tivoli Directory Integrator specific command. The Execute Command can be used when a user configures a rule to execute commands that are specific to the target computer or to execute Tivoli Directory Integrator commands that are not exposed by AMC. For example, in AMC we do not have actions that can restart a server or load a config. The user has to perform

the restart or reload commands using either the TDI Server or Config Files windows. If any error occurs while executing the command, it is captured in the %ACTION_ERROR% variable, which can be further used by the Action Manager.

Target Computer Name

Name or IP address of the target computer. Action Manager connects to the computer specified in this field. If neither a computer hostname nor an IP address is specified, the command executes on the computer where the Action Manager is running.

Port Port specifies the channel over which the Action Manager can connect to the target computer where the command is to be executed.

Username

The user name is verified for authentication and authorization when establishing a connection with the target computer.

Password

The password is verified for authentication and authorization when establishing a connection with the target computer.

Keystore

Keystore path is entered and used in case certificate authentication is required when connecting to the target computer.

Keystore Password

Keystore password is required when certificate authentication is mandatory for connection to the target computer.

Protocol

The protocol that is to be used for establishing a connection with the remote machine. Protocol can have the following values, WINDOWS, RSH, SSH OR REXEC (Windows, remote shell, secure shell, or remote execution protocols).

Command

Command that is to be executed.

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 Solution View. To add this action to the rule, select **Notify event**. 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 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 "Solution View Details" window 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.

Send Email

This action causes an email to be sent to the recipient you specify. You supply the content of the email. Along with the content, the Action Manager provides other details before sending the mail. In the content input area as well as in the subject line, you can specify the variable %EVENT_DATA% value. Specifying %EventData% inserts the actual value of the Eventdata variable when the mail is sent. %Action_Error% can also similarly be substituted here. If **Attach Action Manager Log** is enabled, the Action Manager logs (as specified in the am_logging.properties file) are sent as an email attachment. In the content input area, you can specify the variable %EVENT_DATA% value. Specifying %EventData% in the content puts the actual value of the Eventdata variable when the mail is sent. %Action_Error% is also similarly be substituted here. If **Attach Action Manager Log** is enabled, the Action Manager logs (as specified in the am_logging.properties file) are sent as an email attachment.

Substitute variable for event data: Select **Action Manager** from the left navigation pane or select **Action Manager** from the **Welcome** screen. Under Action Manager, you can Add a rule to a Solution View. You can name a new rule, and edit or delete an existing rule. You can make Event Data available when configuring or sending that data to other actions.

Use Action Manager to make event data available when configuring actions for a trigger. In Action Manager, you can **Add**, **Modify**, or **Delete** a rule. When you add a rule, you name the rule and select the **Trigger type**. Variable substitution allows you to select data that is output from certain Action Manager triggers, and use that data in certain actions that are triggered by a rule. AMC and Action Manager make the data available to the triggered actions in the form of a reserved variable. The action then uses the data that is stored in the variable. You can use this reserved variable in any of the actions you have configured for this trigger.

Triggers that can produce event data: The following trigger types can produce event data that can be consumed by actions:

- **On Start AssemblyLine** - Event data is available as %Event_Data%.
- **On AssemblyLine Terminate** - Event data from On AssemblyLine Terminate is available as %Event_Data%.
- **On Received Event** - Event data from the received event is mapped as %Event_Data%.
- **On Local Variable** - Event data from the Local variable event is mapped as %Event_Data%.
- **On Config Load** - Event data from the On Config Load event is available as %Event_Data%.
- **On Config Unload** - Event data from the trigger would be available as %Event_Data%.
- **On query AssemblyLine result** - Event data is available as %attribute_name%. The %attribute_name% variable is replaced with the details about the actual attribute from the last work entry.
- **Inspect AssemblyLine Exit Code** - Event data is available as %attribute_name% and %Event_Data%.
 - **Inspect Error Object set to enabled** - While configuring the Inspect AssemblyLine exit code trigger, if the user enables Inspect Error Object (sets the option to true), the %Event_Data% variable is replaced with actual error data. The %attribute_name% variable is not available for actions.
 - **Inspect Error Object set to disabled** - While configuring the Inspect AssemblyLine exit code trigger, if the user sets the Inspect Error object to disabled (sets the option to false), the %attribute_name% variable is replaced with the details about the actual attribute from the last work entry. The %Event_Data% variable is not be available for actions.

Actions that can access event data: The actions executed for each of the above triggers can access the event data produced by the triggers using the %Event_Data% variable. Every occurrence of %Event_Data% is replaced with the actual event data for that trigger. The following action types can use event data available from their respective triggers:

- **Notify Event** - Users can specify the %Event_Data% variable in the Data text field only.
- **Write to log** - Users can see a log message that is logged to a database. If the log message, after substitution for the %Event_Data% variable, exceeds 500 characters, the log message is truncated to the first 500 characters. This is because the database has a limit of 500 characters only.
- **Send E-mail** - Any event data specified by %Event_Data% or error data specified by %Action_Error% is substituted in the subject line of the email. Action Manager appends other data about execution before sending the mail. You can specify the variable %EVENT_DATA% value in the content textbox. Specifying %EventData% in the content substitutes the actual value of the Eventdata variable when the mail is sent. You can also similarly substitute %Action_Error% here. If **Attach Action Manager Log** is enabled, the Action Manager logs (as specified in the am_logging.properties file) are sent as an email attachment.

For any action being executed, such as the Send E-mail action, the Execute command action, the Log action, the Start AssemblyLine action, and so on, executes in response to the same trigger, the string of %Event_Data% automatically gets replaced by the event data generated by that trigger.

View Rules Summary: To view the current Action Manager for the selected AssemblyLine, click **View Rules Summary**. The table lists all the defined rules, triggers and actions associated with the Solution View. When you are done viewing, click **Close**. Only those rules which are in Enabled state are listed here.

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 **Advanced** → **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 is 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. These read-only properties are shown in the respective Property Stores. Trying to modify these properties has no effect.

Select Solution View

This window allows you select a Solution View. The menu contains only those Solution 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 Solution Views being created. Once you have selected a view, click **Set**.

After you select a Solution View, you can manage properties by clicking on the other property tabs, such as **Solution Properties** and **Global Properties**.

Solution Properties

This window allows you to add, edit and delete properties in the Solution Properties list.

Global Properties

This window allows you to add, edit and delete Global Properties.

Java Properties

This window allows you to add, edit and delete Java properties.

System Properties

This window allows you to add, edit and delete System properties.

Password Store

This window allows you to add, edit and delete properties in the Password Store.

User Property Store

This window allows you to add, edit and delete properties in the User 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. Select the property store whose associated properties you wish to view, add, edit or delete.

Log Management

If you have not done so already, expand the **Advanced** 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 **Log Management**. When you select a new Solution View, you can click **Refresh**. Clicking **Refresh** lists all of the AssemblyLines that belong to the Solution View you just selected.

This window allows you to select the name of a Solution View. The AssemblyLines listed for deletion are taken from the Solution View you choose. 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 manage display and deletion of logs:

1. Select the Solution View with the AssemblyLines whose logs you want to clean up from the **Solution View** menu.
2. In the **Choose Component** section, do one of the following:
 - Select the **All AssemblyLines** radio button to delete the logs of all AssemblyLines within the selected Solution View.

- Select the **Specific assembly line** radio button to delete only those logs associated with a specific AssemblyLine.
3. If you selected **Specific assembly line**, select the AssemblyLine with logs you want to delete from the menu.
 4. In the **Display Log Files** section, do one of the following:
 - To delete all logs belonging to the selected AssemblyLine(s), select **All**.
 - To delete logs within a date range, use the **Start Date** and **End Date** options. Logs created within the two dates specified will be deleted. Enter the desired dates 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.
 - For pre-Tivoli Directory Integrator 7.1.1 servers, to delete logs older than a certain date, select the **End Date** option. All logs older than the date specified are deleted.
 - To preserve your most recent logs, select the **Display first** radio button. Enter the number of recent logs you want to save. Used to specify that keep those log files that are recent and list the others. You draw the line on *recent* using the edit box. If you type 20, you are telling AMC to keep the most recent 20 log files and list the rest of the files in the table so that they are available for deletion. If you type the number 10, the 10 most recent logs are saved.
 5. In the **Logfiles** table, a list of log files displays. In the **Select** column, select any logs you want to delete and click **Delete**. In the **Select Action** menu, you can choose any of the following options:
 - Export data
 - Change all selected
 - Collapse table
 - Restore
 - When you have selected one of these options, click **Go**.
 6. From the display that results from the criteria you have selected, choose the logs you want to delete and click **Delete** to remove the specified logs. When you are finished deleting logs, click **Close** to exit this window.

Preferred Solution Views

You can select the Solution Views that you want to be loaded by default in the monitor window, using the **Preferred Solution Views** panel.

The "preferred" Configs are shown by default in the monitor status page when it is opened. If there are no views defined then this panel will simply display a message saying that there are no views. Once a set of views are defined then a user can set what he would like to see as the default views. This panel can be viewed by any user who has a set of views assigned to him by the superadmin.

You can make a Solution View preferred by selecting its checkbox in the **Select** column, and click **Enable As Preferred**.

Conversely, you can disable the Preferred status for a Solution View by selecting its checkbox in the **Select** column, and click **Disable As Preferred**.

AMC and AM Command line utilities

A number of command line utilities are included with AMC and its associate product, the Action Manager (AM). These command line utilities help in installing, uninstalling or re-installing the AMC war file. There are also scripts for backup and restore, as well as a migration script. The migration script is for migrating to future versions of AMC and AM, and not for migrating from previous version to the current version. All these scripts get installed in the *TDI_install_dir/bin/amc* directory.

install The install.bat (.sh) script is used to deploy the AMC console module on ISC SE or Tivoli Integrated Portal (ISC embedded). The script relies on the setupCmdLine script for setting up the

necessary environment variables and the tdiISCHome script for determining the location of the ISC runtime and also the type of runtime being used, that is, whether it is the embedded Web platform or WAS. This script is called by the installer.

Usage: install

This script does not take any parameters.

uninstall

The uninstall.bat (.sh) script uninstalls the AMC console module from ISC SE or Tivoli Integrated Portal (ISC embedded). The script relies on the setupCmdLine script for setting up the necessary environment variables and the tdiISCHome script for determining the location of the ISC runtime and also the type of runtime being used, that is, whether it is the embedded Web platform or WAS.

Usage: uninstall

This script does not take any parameters.

backupamc

The backupamc.bat (.sh) script backups all the configuration related information of AMC (configuration files, logs, and so forth.) A backup_tdiamic folder will be created inside the backup directory.

Usage: backupamc [-d *folder_to_create_backup_in*]

If the -d option is not specified, the files are copied to the *TDI_install_dir/bin/amc/ActionManager/backup_tdiamic* directory.

The following files are backed up:

1. amc.properties
2. logging.properties
3. amcdbschema.xml
4. amcdbhandler.properties

restoreamc

The restoreamc.bat (.sh)script will restore the backed up files to a fresh AMC deployment. The backed up files need to be first obtained by using the backupamc script for this to work.

Usage: restoreamc

This script does not take any parameters.

migrateamc

This provides a single backup, restore, uninstall and install command. This will backup the old AMC data, uninstall the old AMC plug-in archive, installs the new AMC and restores the old AMC configuration data.

This script requires the new AMC plug-in archive to be copied into the *TDI_install_dir/amc* directory.

Usage: migrateamc.bat [-d *backup_directory*]

start_tdiamic

This script is a convenient wrapper utility to start AMC. This script internally starts the ISC runtime. If the runtime is the embedded Web platform then it calls the lwiStart command, else if the runtime is WAS then it calls the startServer server1 command. Before starting the ISC

runtime the script calls the `startNetworkServer` command, which is used for starting the Derby database in secured network mode. If the database type is anything other than Derby, then this script only starts the ISC Runtime.

On Windows platforms:

Usage: `start_tdiamc [Service name]`

If a service name is passed, the service will be started instead of calling `lwiStart`.

On Unix Platforms:

Usage: `start_tdiamc`

stop_tdiamc

This script is a convenient wrapper utility to stop AMC. This script internally stops the ISC runtime. If the runtime is the embedded Web platform then it calls the `lwiStop` command, else if the runtime is WAS then it calls the `stopServer server1` command. After executing the command the script makes a call to the `stopNetworkServer` script to stop the Derby database. If the database type is anything other than Derby this script only stops the ISC Runtime.

On Windows platforms:

Usage: `stop_tdiamc [Service name]`

If a service name is passed, the service will be stopped instead of calling `lwiStop`.

On Unix Platforms:

Usage: `stop_tdiamc`

startAM

The Action Manager is started using the `startAM.bat(sh)` script located in the `TDI_install_dir/bin/amc` directory.

Note: The script has the Classpath defined for all the jars required by the Action Manager. There are two variables, `CLASSPATH` and `DB_CLASSPATH`. The `DB_CLASSPATH` has the path separated list of .jar files required for achieving JDBC Connectivity with the database. When AMC is configured to use Oracle, MS SQL Server or DB2 the corresponding JDBC .jar files of these databases should be added to the `DB_CLASSPATH` variable.

On Windows, the script accepts an optional service name parameter that can be used to start an already registered service:

`startAM.bat [service name]`

stopAM

The Action Manager is stopped using the `stopAM.bat(sh)` located in the `TDI_install_dir/bin/amc` directory. This script uses the `processID` of the started AM to kill it. The `processID` is obtained by the `startAM` script and is stored in a file, which in turn is read by the `stopAM` script.

On Windows, the script accepts an optional service name parameter that can be used to stop an already registered service:

`stopAM.bat [service name]`

startNetworkServer

This script is used for starting the Derby database server in network mode, on port 1528. The port selected is different from the default port of Derby.

Usage: `startNetworkServer`

stopNetworkServer

This script is used for stopping the Derby database server in network mode.

Usage: `stopNetworkServer`

setDBType

This script is used for setting the type of database that you are using. The script sets the property

namely DB_TYPE. If the DB_TYPE is set to derby then on executing the startNetworkServer script the Derby database will be started on the host and port that you specified in the startNetworkServer script file. The setDBType also sets the database user name and password. The database user name and password are required by startNetworkServer to enable the BUILTIN security mechanism and to add the user to the list of authorized users.

The setDBType script is called internally by the startNetworkServer and stopNetworkServer scripts for setting the DB_TYPE and the DB_USER and DB_PASSWORD properties.

backupamcdb

This script is used during the migration of an AMC database. The script backs up the AMC database and has the data exported in a Tivoli Directory Integrator defined XML format. This script is called by the installer when you choose the migration path.

Usage: backupamcdb -d *folder which contains AMC backup* -p *location of the amc.properties file*

restoreamcdb

This script is used to restore the AMC database during migration. The scripts are called by the installer when you choose the migration path.

Usage: restoreamcdb -d *folder which contains AMC backup* -p *location of the amc.properties file*

backupam

This script is used for backing up the Action Manager properties files. The script backs up the am_config.properties and am_logging.properties file.

Usage: backupam [-d backup_directory]

The archived info is created in the backup folder. If the -d option is not specified the files are copied to the *TDI_install_dir/bin/amc/ActionManager/backup_tdiacm* directory.

restoream

This script is used for restoring the properties files of Action Manager which were backed up using the backupam script. The restore script restores the am_config.properties and am_logging.properties files.

Usage: restoream [-d backup_directory]

If the -d option is not specified, the files are copied from the *TDI_install_dir/bin/amc/ActionManager/backup_tdiacm* directory.

setAMCRoles

This script is used for mapping the user who is installing TDI AMC to the ISC admin and TDI AMC Admin roles. This script is introduced in TDI 7.0.

Once these roles are granted to the install user, that user has the authorization to add new users and grant them with the necessary roles. The install user becomes the administrator for the AMC console module.

Usage: setAMCRole *username* [*OS Group*]

The OS Group is an optional parameter while deploying AMC on ISC SE.

tdimigam

This script is used for migration of the am_config.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
```

Logging for this command is controlled by the tdimigam-Log4J.properties file.

tdimigamc

This script is used for migration of the amc.properties file. The options of this script are similar to those of tdimigam and tdimiggbl that are used for migration of the am_config.properties and global.properties files respectively.

The usage for this command is:

```
tdimigamc -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
```

Logging for this command is controlled by the tdimigamc-Log4J.properties file.

addAMCService

This script is used for adding AMC as a service on a system.

Usage: addAMCService *Service_Name*

On Windows the script registers the Generic Windows Service executable (*TDI_install_dir/bin/amc/amcwinservice.exe*) from the IBM Platform Integration Toolkit. The Generic Windows Service uses the configuration file *TDI_install_dir/bin/amc/amcwinservice.ini*. That file specifies the name of the service and the start/stop commands. The file is automatically populated by the installer or the "addAMCService" script.

By default the file looks like this:

```
[Service]  
ServiceName=$service_name$  
WorkingDirectory="$install_dir$\bin\amc"  
StartCommand=""$install_dir$\bin\amc\amcservice.bat" start amc am"  
StopCommand=""$install_dir$\bin\amc\amcservice.bat" stop amc am"
```

This means that by default the AMC service runs both the AMC and the Action Manager.

After you call "addAMCService", you can edit the .ini file to customize which components are run by the service (both AMC and AM, just AMC or just AM).

For example to run only the AMC, specify start and stop commands like the following:

```
StartCommand=""$install_dir$\bin\amc\amcservice.bat" start amc"  
StopCommand=""$install_dir$\bin\amc\amcservice.bat" stop amc"
```

To start/stop the service use the GUI "Services" utility (**Control Panel -> Administrative Tools -> Services**) or the Service Controller command line tool:

```
sc start <service name>  
sc stop <service name>
```

Beware that in the "Services" utility the display name of the registered service looks like this:

IBM Tivoli Directory Integrator Administration and Monitoring Console – myamc

where "myamc" is the service name that you specified as an argument to "addAMCService.bat".

On UNIX the script appends a line like the following at the end of the /etc/inittab system file:

```
<service name>::once:<install dir>/bin/amc/amcservice.sh start amc am
```

To stop the service use the "amcservice.sh" script from the *TDI_install_dir/bin/amc* folder:

```
amcservice.sh stop amc am
```

or just

```
amcservice.sh stop amc
```

if the service runs only AMC.

deleteAMCService

The script is used for removing AMC as a service from a system.

Usage: `deleteAMCService Service_Name`

setDerbyProps

The script sets the necessary Derby database properties that are used by the `startNetworkServer` and `stopNetworkServer` scripts.

Usage: `setDerbyProps`

amcservice

This script starts/stops the whole Administration and Monitoring configuration. The following configurations are supported:

- both AMC and AM
- only AMC
- only AM

Internally the script calls "start_tdiamc"/"stop_tdiamc" and "startAM"/"stopAM". It is intended to be used when registering an Operating System service.

Usage: `amcservice [start|stop] [amc] [am]`

Examples:

```
amcservice start amc
amcservice stop amc am
```

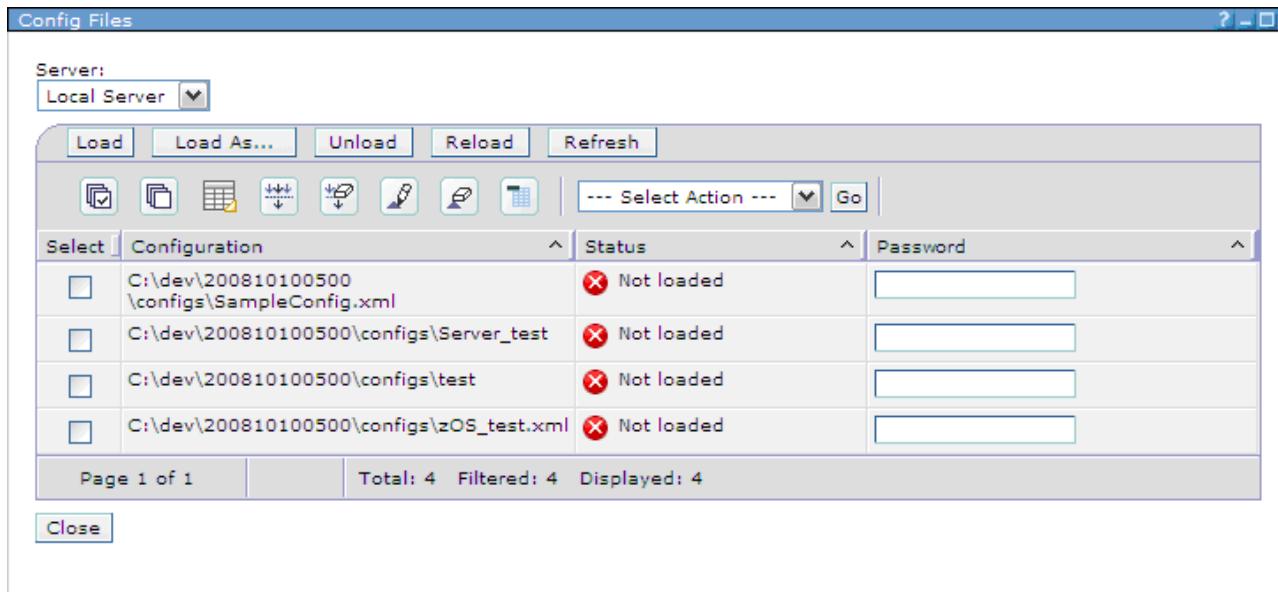
Example walkthrough of creating a Solution View and Rules

This section illustrates all the steps to create a Solution View, configure a rule and trigger it in Action Manager. It is assumed that IBM Tivoli Directory Integrator is installed along with AMC. The configuration `SampleConfig.xml` used in this example is the one that you are supposed to create following the tutorial in the *IBM Tivoli Directory Integrator V7.1.1 Getting Started*, "Introducing IBM Tivoli Directory Integrator" -> "Creating your first AssemblyLine". It should be copied into the *TDI_install_dir/configs* folder, where *TDI_install_dir* is the installation directory of TDI. This solution reads data from the 'examples/Tutorial/People.csv' file and writes it to 'examples/Tutorial/Output.xml'.

This section illustrates all the steps to create a config view, configure rules, and trigger this rule in Action Manager. It is assumed that Tivoli Directory Integrator is installed along with AMC. The sample config (`SampleConfig.xml`) and associated files are available in the downloads section and should be copied into the *TDI_install_dir/configs* folder. This solution reads data from the `sample.csv` file and writes to `sample.xml`.

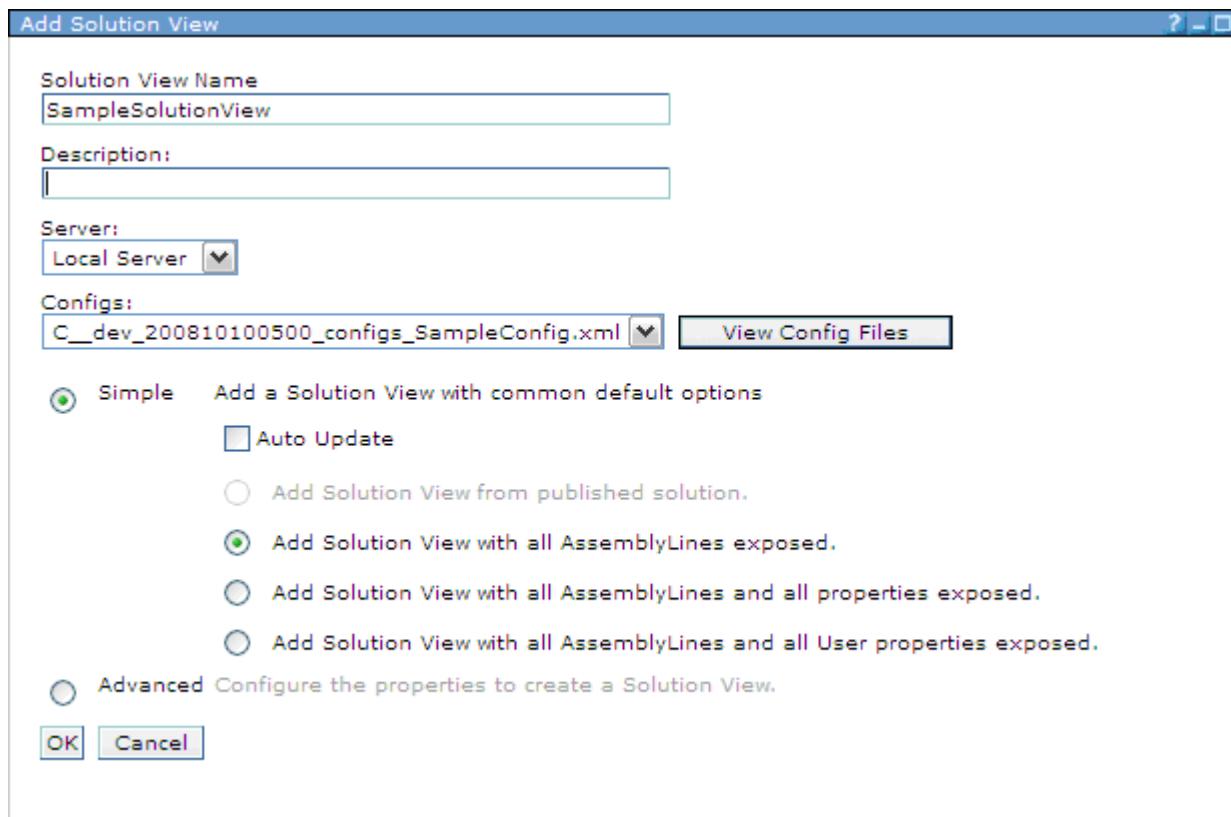
Steps:

1. Start the Tivoli Directory Integrator server in daemon mode.
2. Start AMC using the following command - *TDI_install_dir\bin\amc\start_tdiamc.bat*
3. Logon to the AMC console using the URL with the following syntax - *http://hostname:port/ibm/console* using the default username and password.
4. After logging on to the AMC console, select the Servers on the navigation panel and then choose the Server you wish to use. Press the **Config files** button and the following panel is shown.



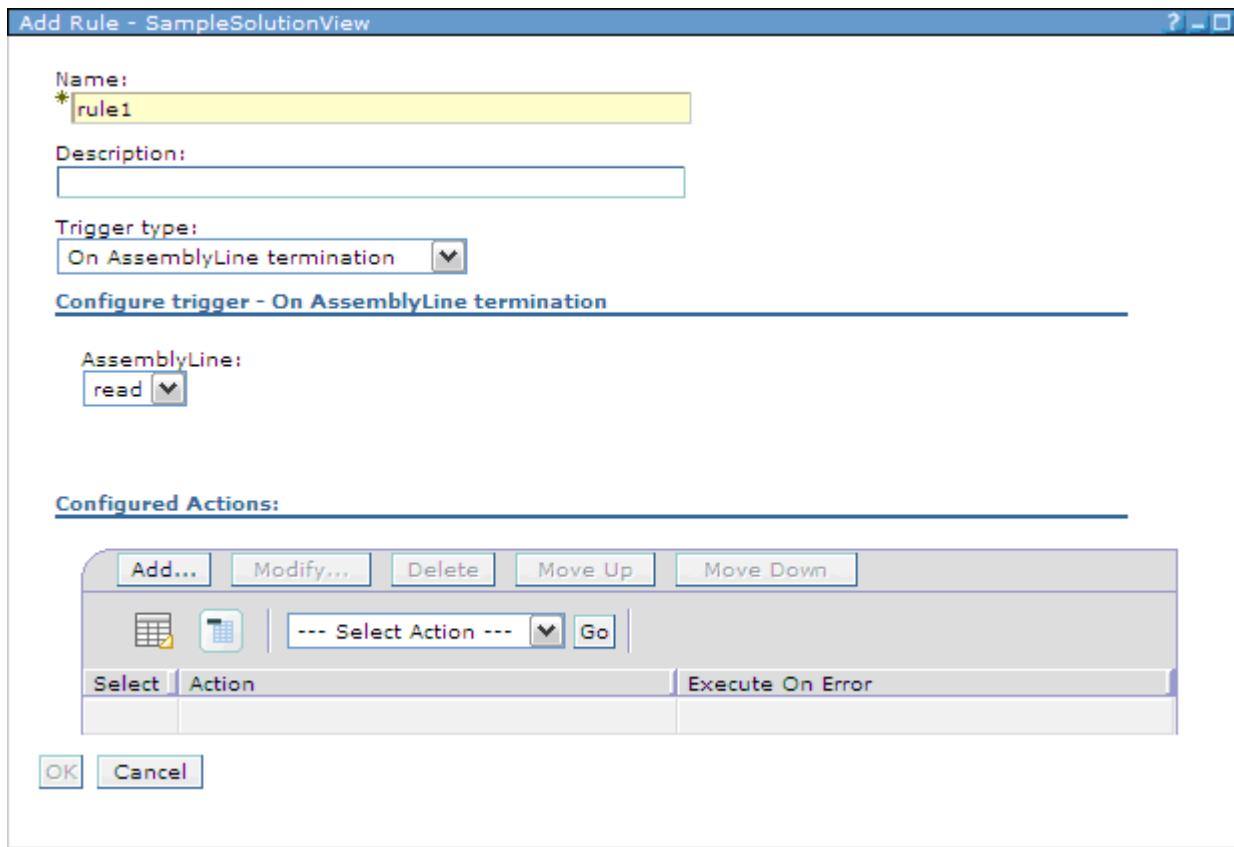
Select the SampleConfig.xml file and click the **Load** button.

5. Select the Solution View link of the navigation panel to create a solution view for the loaded config. Select the Add button and the following panel will be shown.

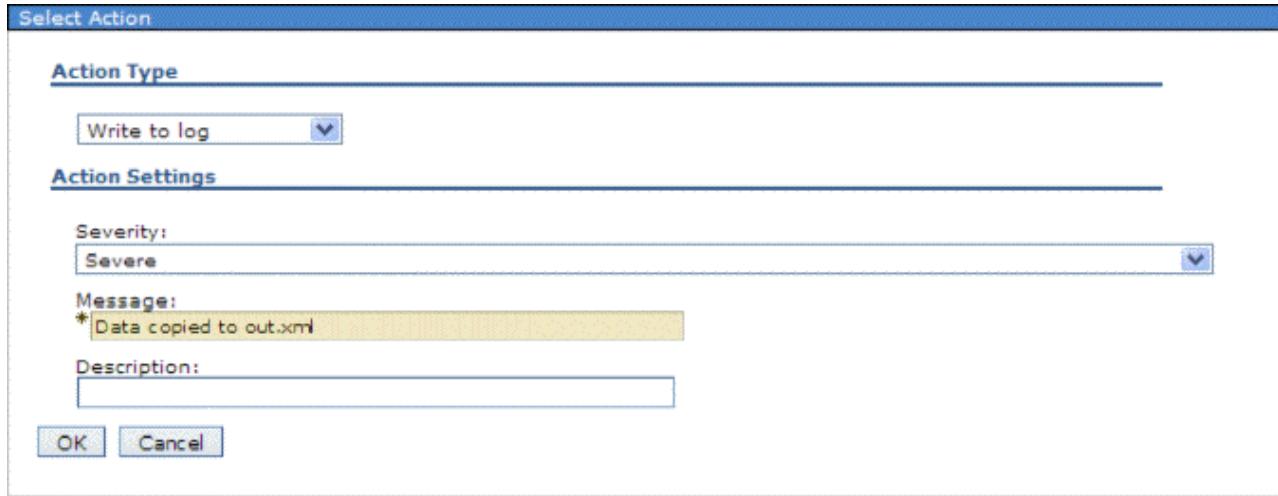


Add a suitable name for the config, for example, SampleSolutionView. On selecting OK, a message indicating that the Solution View SampleSolutionView was created successfully will be shown.

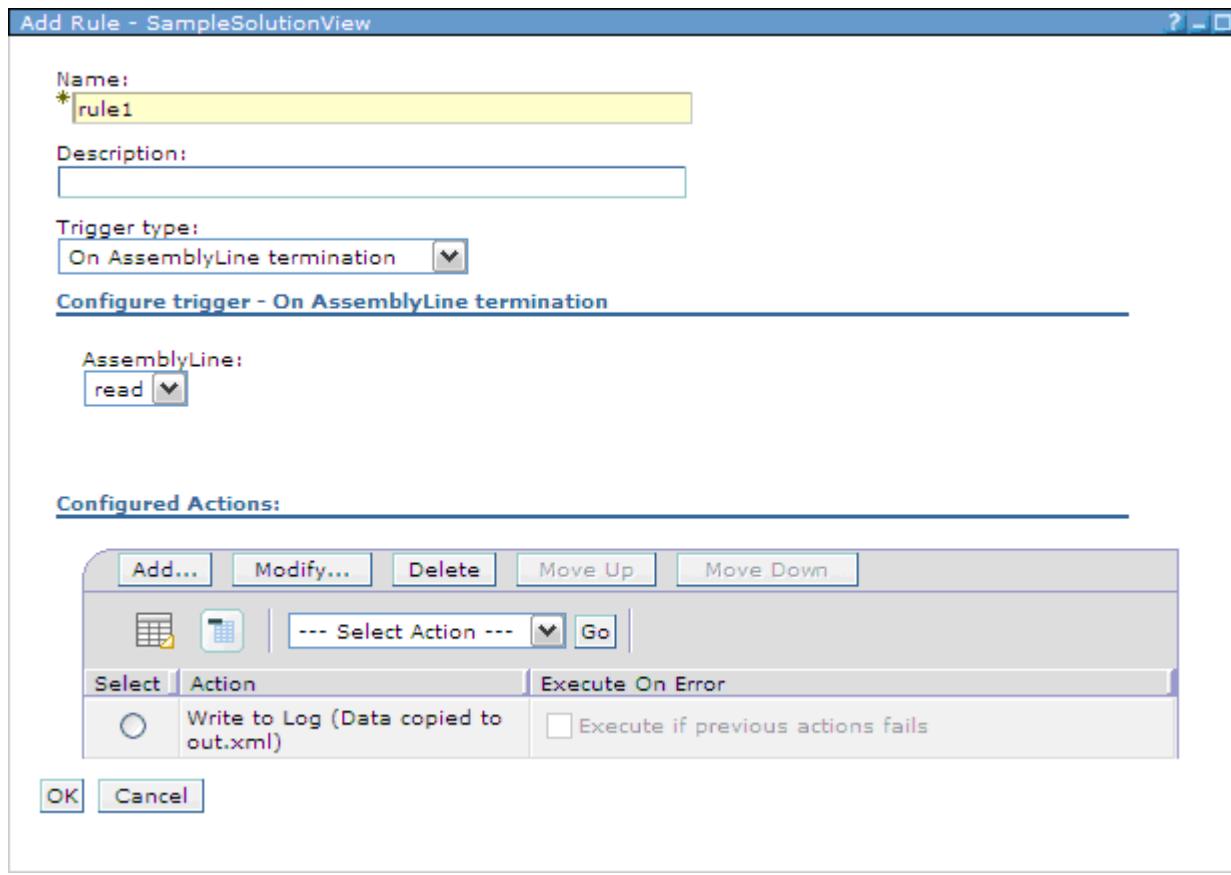
6. Select the Action Manager link on the navigation panel to reach the Action Manager rules configuration screen. Select "SampleConfigView" from the Select solution views dropdown box. When clicking the **Add** button in the Configured Rules section, the following panel will be shown.



Enter a name, for example "rule 1". Select trigger type "On AssemblyLine termination" and click the **Add** button in the Configured Actions section.



In the Select Action panel, select the "Write to log" option in the Action Type combo box. Add the text "data copied to out.xml" in the Message text box and click **OK**. The following rule panel will be shown.

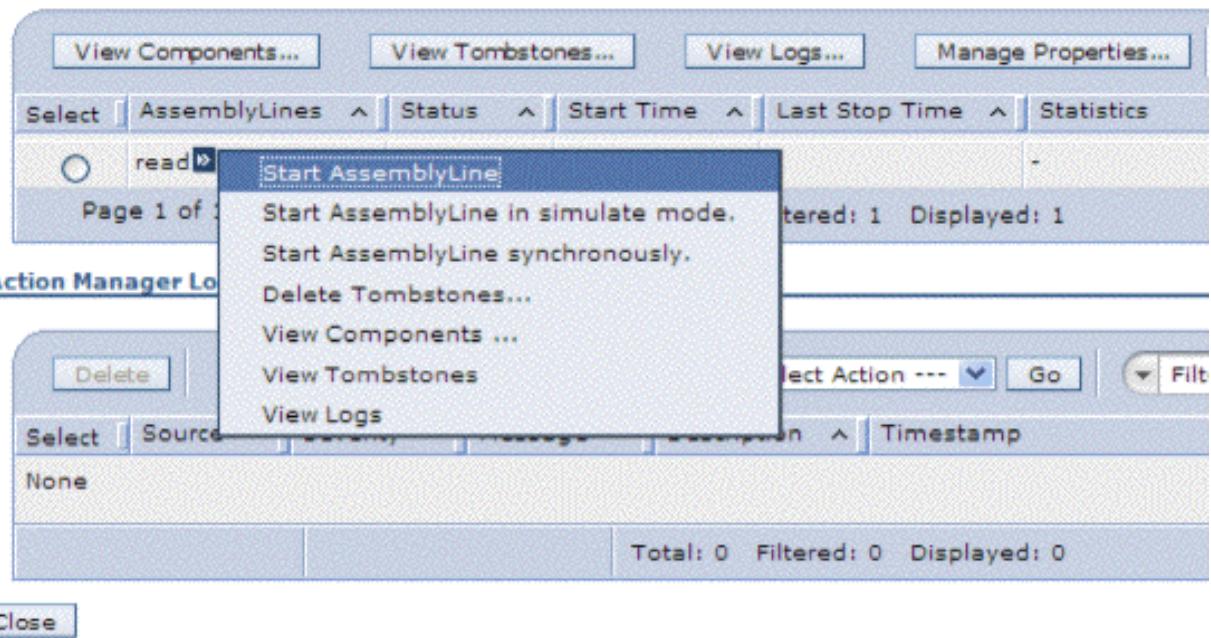


Clicking **OK** completes the creation of this rule and the AMC configuration required for this example.

7. Start Action Manager using *TDI_install_dir\bin\amc\startAM.bat*. A thread is created for the rule "rule1", which should be waiting for the termination of the specified AL.
8. To trigger this rule, the "read" AssemblyLine of SampleConfig.xml needs to be executed. Select **Monitor Status** on the navigation panel. On the Monitor Status panel, select the SampleSolutionView and click on the **Solution View Details** button. The following panel will be shown.

Solution View Details - SampleSolutionView

AssemblyLines



9. Start the AL using the option shown above. The rule will be triggered and the following status will be displayed on the Action Manager console.

```
C:\WINDOWS\system32\cmd.exe - startAM.bat

2008-10-16 17:05:01 com.ibm.di.amc.actionmanager.ServerConfigHandler startAMCServerModificationListenerThread
INFO: CTGDJB659I Started AMCServerModificationListener thread.
2008-10-16 17:05:01 com.ibm.di.amc.actionmanager.api.AMService initialize
INFO: AM.RMI.REGISTRY.SYSTEM.SECURITYMANAGER.NULL
2008-10-16 17:05:01 com.ibm.di.amc.actionmanager.api.AMService initialize
INFO: AM.RMI.REGISTRY.RESETTING.SYSTEM.SECURITYMANAGER.NULL
2008-10-16 17:05:01 com.ibm.di.amc.actionmanager.api.AMService initialize
INFO: CTGDJB720I The Action Manager RMI registry is started on port 13104.
2008-10-16 17:05:01 com.ibm.di.amc.actionmanager.AMHandler startHealthALManagerThread
INFO: CTGDJB512I Started Health AssemblyLine Manager.
2008-10-16 17:05:01 com.ibm.di.amc.actionmanager.AMHandler main
INFO: CTGDJB511I Action Manager initialization completed.
2008-10-16 17:05:01 com.ibm.di.amc.actionmanager.AMHandler startDatabaseModificationListener
INFO: CTGDJB532I Database modification listener started.
2008-10-16 17:06:23 com.ibm.di.amc.actionmanager.AssemblyLineTerminateListener triggerRule
INFO: CTGDJB549I Rule rule1 triggered.
2008-10-16 17:06:24 com.ibm.di.amc.actionmanager.RuleExecutionManager writeToFile
SEVERE: CTGDJB622I [Message]: Data copied to out.xml
[Description]: None

2008-10-16 17:06:24 com.ibm.di.amc.actionmanager.RuleExecutionManager execute
INFO: CTGDJB615I Action successfully executed.
[Solution View]: SampleSolutionView
[Rule]: rule1
[Action Order]: 2
[Action Type]: WRITE_LOG
```

On AMC, the Action Manager logs table on the Solution View Details panel will be shown as below.

Solution View Details - SampleSolutionView

AssemblyLines

View Components...		View Tombstones...		View Logs...		Manage Properties...	
						--- Select Action ---	
Select	AssemblyLines	Status	Start Time	Last Stop Ti...	Statistics		
	read	Stopped	-	-	-		
Page 1 of 1		Total: 1 Filtered: 1 Displayed: 1					

Action Manager Logs

Delete					
Select	Source	Severity	Message	Description	Timestamp
<input type="checkbox"/>	rule1	SEVERE	Data copied to out.xml	None	16.10.2008 17:06:24
Page 1 of 1		Total: 1 Filtered: 1 Displayed: 1			

[Close](#)

Chapter 17. Touchpoint Server

The Touchpoint Server shipped with Tivoli Directory Integrator provides access to Tivoli Directory Integrator components (Connectors and AssemblyLines) through a ReSTful communication protocol. Clients send HTTP requests to the Touchpoint Server in order to request from a Tivoli Directory Integrator Server to create a Touchpoint Instance which the client can then "talk" to.

A Touchpoint Instance is implemented using standard Tivoli Directory Integrator Components and allows HTTP based clients to access third party systems that Tivoli Directory Integrator can "speak" to. In this context the Touchpoint Instance can be thought of as a "proxy" between a client application and a remote service, as it allows clients to use a unified protocol for communication to a variety of systems that don't have an HTTP based interface.

Touchpoint concepts

In essence, the Touchpoint protocol is a provisioning protocol which gives access to Tivoli Directory Integrator Connectors and AssemblyLines through HTTP. When you create a Touchpoint, you get a "proxy" through which to work with a remote system. Once the Touchpoint is created and configured, you only send HTTP requests and you are completely isolated from the specifics of that system.

The sections below provide details about the different concepts associated with Touchpoints:

Touchpoint Server

The Touchpoint Server is the application that stores information about the defined Touchpoints in the particular domain. The Touchpoint Server is used to control the remote Touchpoint instances. It is responsible for their configuration, starting and stopping. The Touchpoint Server is provided as a service running inside the Tivoli Directory Integrator Server.

Clients of the Touchpoint Server are using the Atom Publishing Protocol to access details for the Touchpoint Providers, Touchpoint Types and Touchpoint Instances. For more details on the defined schema see section "Touchpoint Schema" on page 272.

Touchpoint Provider

A Touchpoint Provider is a server on which the Touchpoint Server is creating the Touchpoint Instances. In our case a Touchpoint Provider can be any Tivoli Directory Integrator Server version 7.1 or later. The Touchpoint Server is only able to work with Tivoli Directory Integrator Servers, as any other Touchpoint Provider is not supported.

The Touchpoint Server is shipped as an add-on to the Tivoli Directory Integrator Server. It runs in the server's JVM which enables it to communicate with the server through the Local Server API. This is why the Tivoli Directory Integrator Server has been registered as a local Touchpoint Provider by default. In order to register a Touchpoint Provider, representing a remote Tivoli Directory Integrator Server, you must use the RMI settings of the remote server. The registration for both a Local and a Remote Server is performed in the standard Touchpoint Server configuration file.

Note: No user interface panels to configure the Touchpoint Server or Touchpoint Providers currently exist in the Configuration Editor or the Webadmin tool, AMC. All configuration must be done by means of XML configuration files.

For more details see section "Touchpoint Configuration" on page 276.

Once registered the Touchpoint Provider cannot be changed through the Atom interfaces. Additionally the Atom interface hides some of the details specifying the connection the remote Tivoli Directory Integrator Server. Those details are only used by the Touchpoint Server in order to communicate with the Touchpoint Provider and are not meant to be seen by the clients of the protocol.

Touchpoint Type

A Touchpoint Type is an abstract notation that provides meta-information for each Touchpoint Instance and determines its behavior. Every Touchpoint Instance has exactly one Touchpoint Type, while there is no limit to how many Touchpoint Instances can be created for a particular Type.

There are three categories of Touchpoint Types:

standard

This category corresponds to the Tivoli Directory Integrator Connectors supported by the chosen Touchpoint Provider and start with the prefix `system`. Every Touchpoint Provider has its own set of standard Touchpoint Types, as it provides different Connectors to you. By choosing any of these types, you specify that they will rely on the base Touchpoint Template for the structure of their Touchpoint Instance (for details about Templates see section “Touchpoint Template” on page 267). Furthermore, the chosen Type determines the inheritance of Template's Service Connector (the Connector working with a third-party system). For example, if you need to read data from a RDBMS, the JDBC Connector can be used. Hence, you will create a Touchpoint Instance with Type `system:/Connectors/ibmdi.JDBC` and configure it appropriately. These Touchpoint Types support only the Provider and Initiator Touchpoint roles.

custom

This category represents custom Touchpoint Templates provided by you and are distinguished by the prefix `file`. Instead of relying on the base template that comes with TDI, you can create your own ones, tweaking the Touchpoint behavior according to your needs. In exchange, though, they lose some of the flexibility offered by the base Template. Once a custom Template has been created, the type of its Connectors cannot be changed, limiting the created Touchpoint Instances to always work with the same type of remote system. An example use of the custom Touchpoint Type is if you need to work with several data sources, for example, read data from a RDBMS and add information from an LDAP server. This cannot be achieved by a single Touchpoint Instance relying on the base Touchpoint Template. To solve this, you can create a custom Touchpoint Template and use its corresponding Touchpoint Type (for example, `file:/template_file_name.xml`) for creating a Touchpoint Instance. The only limitation is that all subsequent Touchpoints created from this Type will also work with a RDBMS and an LDAP server (since the type of the Service Connector cannot be changed). These Types support all Touchpoint roles.

virtual

This category consists of only one Touchpoint Type with name `virtual://Intermediary`. Unlike the above Types, which are connected to some actual resource (a Tivoli Directory Integrator Connector for standard Types and a Template file for custom Types), this Touchpoint Type is used for providing a way to create an Intermediary Touchpoint Instance out-of-the-box. For this purpose it relies on the base Touchpoint Template provided with Tivoli Directory Integrator. This Touchpoint Type supports only the Intermediary Touchpoint role.

When creating a Touchpoint Instance, you need to provide the configuration of the Connector that will communicate to the third-party system. If a standard Touchpoint Type is used, this means providing the configuration of the corresponding Tivoli Directory Integrator Connector. For custom Types you must provide configuration for the Service Connector of the custom Template. Finally, for virtual Types no configuration is needed as Intermediary Touchpoint Instances rely only on HTTP Components for performing their task.

How do you find out what parameters are needed to create a Touchpoint Instance? For this purpose the Touchpoint Server supports Property Sheet Definitions – XML documents providing information for the

schema of a TDI Component. To obtain the URL of the Property Sheet Definition, send a HTTP GET request to the URL of the particular Touchpoint Type. It defines the required connector parameters, their default values, as well as other useful information needed when configuring a Touchpoint Instance.

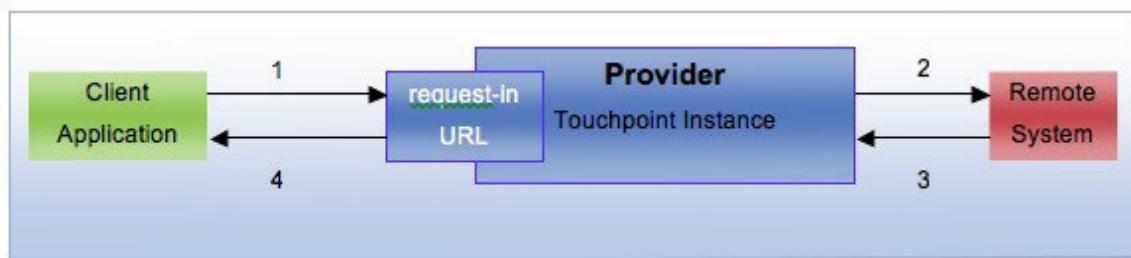
In general, the information provided by Property Sheet Definitions is similar to the one available when configuring a Connector in TDI's Config Editor (except for parameter descriptions). For more details on Property Sheet Definitions and their usage see section "Property sheet definitions" on page 281.

Touchpoint Instance

By nature, a Touchpoint Instance represents a proxy that enables access to a remote service over the common HTTP protocol. However, the communication flow varies significantly, depending on the role of the Touchpoint Instance. Here are more details for the supported Touchpoint roles:

Provider

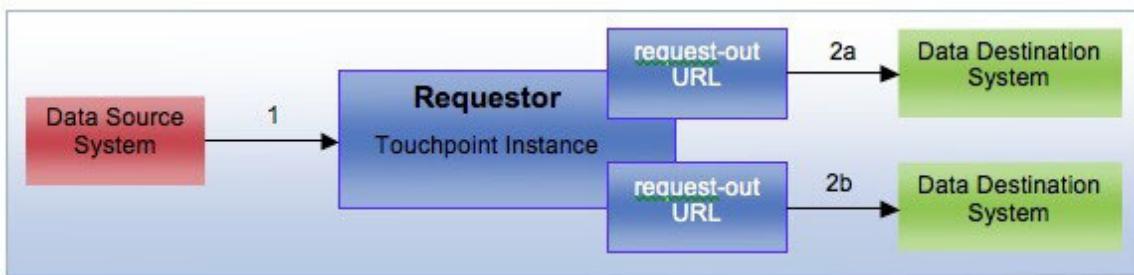
This mode provides access of clients to third party services. A client sends an HTTP request to the Touchpoint Instance (1), which in turn translates the request to the native language of the remote system and sends that to it (2). The remote system responds with the proper result to the Touchpoint Instance (3), which then transmits it back to the client (4) through HTTP.



As can be seen from this diagram, the Provider has a **single input interface**, represented by a URL on which you can send your requests (request-in URL). This URL is intrinsic, meaning that the Touchpoint Instance creates it and provides it to you.

Initiator

This mode provides transport of information between two ends. The Touchpoint Instance is the one that requests a piece of data from one system using the system dependent language (1) and then "pushes" this data through HTTP to several Data Destination System (2a, 2b) - HTTP servers capable of receiving data (for example, a Provider Touchpoint).



To achieve this, the Initiator is allowed to have **multiple output interfaces** – request-out URLs where the available data is sent. These destination points (also referred to as Destinations) can be added or removed from the Initiator at runtime, which gives a lot of flexibility for distributing data between systems. The Initiator starts working when its first request-out URL is added and stops when all of them are removed. By default the Initiator Touchpoint sends data to all of its Destinations, disregarding their responses. This behavior can be modified (for example, to stop the Initiator, if any of the Destinations fails to receive the data) by editing the base Template or providing a custom one.

Intermediary

This mode provides a forwarding service. The Intermediary Touchpoint Instance accepts requests (1) and then sends them to several Destinations through HTTP (2a, 2b). The Destinations respond to the Touchpoint Instance (3a, 3b), it merges their responses and transmits the result back to the caller (4).

For a Client Application, the Intermediary looks like a Provider giving access to some third party system, while for the Data Destination Systems it is an Initiator sending data.



The Intermediary Touchpoint has a **single input interface** – request-in URL and **multiple output ones** – request-out URLs. You configure the output interfaces the same way as for the Initiator role, and the input interface is similar to the Provider's one (meaning its URL is set by the Touchpoint Server). In its simplest form, the Intermediary Touchpoint does not modify the forwarded data but you may provide such logic by editing the base Touchpoint Template or by providing a custom one.

Another specific characteristic of the Intermediary Touchpoint can be noticed when it acts as a proxy for accessing several Providers. As mentioned above, you can interact with this complex system as if it was a simple Provider. However, there is an explicit need to merge the responses returned from the end Providers. The default available logic is:

- if one of the Destinations returns a successful response, it is returned to the caller;
- if several Destinations return a successful response, the data in the response bodies is merged and returned with an HTTP code 200;
- if all Destinations return failure responses, an error response with code 500 is returned to the caller. In its HTTP body, it will contain the following information for each Destination:
 - Request to URL 'the URL of the Destination' failed.
 - HTTP status: the returned HTTP error code
 - HTTP body: the returned HTTP body

To change the merging behavior you need to edit the Touchpoint Template used.

For more details on the communication protocol between the Client and the Touchpoint, see section “Touchpoint Instance communication protocol” on page 278.

Besides the above mentioned roles you must specify two more configuration items when setting up a Touchpoint Instance. These are:

- **Property Sheet** of the Touchpoint. Each Touchpoint Instance is representing a specific Touchpoint Type, which means that every Touchpoint Instance has a specific configuration. It complies with the one defined by the Touchpoint Type (the Property Sheet Definition) and reflects the schema of the TDI Connector working with the remote system for this Touchpoint (known as the Service Connector).

Note: The Intermediary Touchpoint does not use a Service Connector for its working, so no such configuration information is needed for it. When setting up such Touchpoints, users should send an empty property sheet, as any provided parameters will just be ignored.

For more information about the Property Sheets format see section “Instance Configuration” on page 276.

- **Admin state** of the Touchpoint. This item is used for micro managing the state of the Instance. For example, you may want to disable a running Provider Touchpoint for a given period of time. Instead of deleting it, you can simply set its admin state to disabled. This way, when it is needed again, you only have to update its state to enabled, and it will be running again.

Another possible use case is for the Intermediary and Initiator roles. As soon as you add the first Destination to such Touchpoints, they start working (and presumably sending data). Additional Destinations can be added later on, but this may lead to data lost as some of the data will already be sent. To solve this, you can create such Touchpoints with disabled admin state (preventing them from starting) and add as many Destinations as needed. When the configuration is done, you can change the state to enabled, and the Touchpoint will start sending data to all the Destinations.

After creating the Touchpoint Instance you can access its current Operational state. There are two different states a Touchpoint can be in and their meaning varies with the role of the Instance.

A *Provider Touchpoint Instance* has the following states:

- **Unavailable** – when the Touchpoint Instance is intentionally disabled through its Admin state.
- **Available** – when the Touchpoint is configured and its admin state is enabled.

The status of a Provider Touchpoint has one additional parameter. Besides the operational state, you can get the request-in of the Touchpoint – the URL address where you send your request so that the Provider Touchpoint can communicate them to the remote system.

An *Initiator Touchpoint Instance* has the following states:

- **Unavailable** – the Touchpoint Instance is in this state if:
 - it is not fully configured, meaning that it has no Destinations (request-out URLs);
 - it is intentionally disabled, by setting its Admin state;
 - its Service Connector has finished reading from the data source.
 This specific case is due to the behavior of the Service Connector of the Initiator Touchpoint Instance. If this Connector is a standard Iterator, it will read its configured data source and when done will stop, thus stopping the whole Touchpoint. However, in the case of Change Detection or JMS Connectors, it will continue to wait for new data and the Touchpoint Instance will keep running indefinitely (and staying Available). In this case it should be stopped explicitly by deleting it or setting its Admin state to disabled.
- **Available** – when in this state, the Touchpoint Instance is actually reading data from the data source.

As mentioned above, the *Intermediary Touchpoint Instance* in essence is a combination of a Provider and an Initiator Touchpoint Instance. This is reflected on its status as well.

- **Unavailable** – the Touchpoint Instance is in this state if:
 - it is not fully configured, meaning that it has no Destinations (request-out URLs);
 - it is intentionally disabled, by setting its Admin state;
- **Available** – when in this state, the Touchpoint admin state is enabled and it is actively forwarding its incoming requests to the Destinations.

Similarly to the Provider Touchpoint, the Intermediary has a request-in URL, which can be obtained from its Status Entry.

Touchpoint Template

When a Touchpoint Instance is started, the Touchpoint Server creates a Tivoli Directory Integrator configuration and starts it as a temporary Tivoli Directory Integrator ConfigInstance on its associated Touchpoint Provider (TDI Server). The TDI Configuration is based on a base Template provided with the Touchpoint Server. The path to this template file is specified in the Touchpoint Server configuration, the default being `TDI_install_dir/etc/TouchpointTemplate.xml`. It is placed on the Touchpoint Server side

and is a general template not associated with a particular TDI Server. The configuration of each Touchpoint Instance is the one that is filled into the base template before it is started as a ConfigInstance.

The default base template contains the following structure:

AssemblyLines:

- **ProviderServer** – this is the AssemblyLine responsible for receiving HTTP requests and starting the appropriate handler ALs based on those requests. It acts as a common entry point for accessing all of the Provider and Intermediary Touchpoint Instances managed by the Touchpoint Server.

Using this technique, only one HTTP Server Connector is needed for communicating with all of these Touchpoints which mean only one TCP port should be opened (by default this is 1097). This avoids possible firewall problems that can occur if a large amount of random ports needs to be opened.

The ProviderServer AssemblyLine uses the following Tivoli Directory Integrator Components:

- *HttpServer* – the Connector receiving the requests from client applications. By default it listens on port 1097, though this can be changed from the configuration file of the Touchpoint Server.
- *HandleRequest* – a TDI Script Component that determines which AssemblyLine should be started depending on the request.
- **ProviderHandler** – this AssemblyLine is started by the ProviderServer when a request for the Provider Touchpoint Instance is received. It performs the actual communication with the remote system. The Components used are:
 - *ServiceConnector* – this is the Connector that communicates with the remote system. It is set to Passive state, which means that it is controlled from a script instead of the AssemblyLine flow. An important characteristic of this Connector is that it inherits from the GenericServiceConnector in the library of the base template (Inherit from: /Connectors/GenericServiceConnector). The role of this parent Connector is described below.

Note: A Touchpoint AssemblyLine should have only one Service Connector (or none, as is the case with Intermediary). If several are provided, they all will have the same configuration.

- *HandleRequest* – this Script uses a servlet-like structure to handle the incoming requests. It initializes the ServiceConnector and depending on the received request performs a different operation over the data source. For more details on the supported Provider operations see section “Touchpoint Instance communication protocol” on page 278.
- **IntermediaryHandler** – when the ProviderServer AL receives a request for an Intermediary Touchpoint Instance it redirects it to this AssemblyLine. Since by default the Intermediary just forwards data to several destinations, it needs only one Component:
 - *SendToDestinations* – this Script Component forwards every request (passed to the AL in the form of a work Entry) to the configured Touchpoint Destinations using the call:

```
sendToDestinations(work, mergeResponsesCallback)
```

This method is provided by the Sender Script Component located in the library of the base Template. Besides the entry that will be sent, it requires a callback function that is used to merge the responses from each Destination. See section “Touchpoint Instance” on page 265 for details on the default merging behavior of the Intermediary Touchpoint.

- **Initiator** – This AssemblyLine is used to represent an Initiator Touchpoint Instance - the only Touchpoint role that is not accessed through the ProviderServer entry point. This AssemblyLine consists of the following Components:

- *ServiceConnector* – this is the Connector used to feed the AL with data from the remote system. As with the ProviderHandler AssemblyLine, this Connector inherits from the GenericServiceConnector in the library of the base Template.

Note: Touchpoint AssemblyLines should have only one Service Connector (or none, as is the case with Intermediary). If several are provided, they all will have the same configuration.

- *ConvertToHTTPContent* – a Script Component that transforms the data read from the Service Connector to an HTTP entry.
- *SendToDestinations* – this Script Component forwards the received request to the Touchpoint Destinations using the call:

```
sendToDestinations(work, null)
```

The same call is used by the Intermediary Touchpoint Instance. The only difference is that here no callback function for merging the responses from the different Destinations is provided. See section “Touchpoint Instance” on page 265 for details on the default merging behavior of the Initiator.

Resources (library of the base Template):

- **Connectors** – several Connectors performing specific tasks for the various AssemblyLines:
 - *GenericServiceConnector* – this Connector is the parent of all Service Connectors. When a Touchpoint Instance is being created, the Touchpoint Server configures this Connector to work with the specific remote system. Since the Service Connector in each AssemblyLine inherits from it, they all get the same configuration.

Notes:

1. The name of the Service Connector in the AssemblyLine is not important provided that it inherits from the GenericServiceConnector.
2. Only one Service Connector per AssemblyLine should be provided.

The Touchpoint Server handles the GenericServiceConnector differently, depending on the Touchpoint Type:

- When using a **standard** Touchpoint Type the GenericServiceConnector is used for setting both the inheritance and parameters of the Service Connectors. For example, consider creating a Provider Touchpoint Instance working with a RDBMS. You create this Instance from Type `system:/Connectors/ibmdi.JDBC`, configure it in Provider mode and pass the parameters needed by the JDBC Connector. This means that the inheritance of the GenericServiceConnector will be overridden to `system://Connectors/ibmdi.JDBC` and the provided JDBC parameters will be set to it. This way, both ServiceConnector-s in the ProviderHandler and Initiator AssemblyLines will get the same configuration as well (since they inherit from the GenericServiceConnector). The ProviderHandler AssemblyLine will be started and you get a Provider Touchpoint Instance for working with a RDBMS.
- When using a **custom** Touchpoint Type the GenericServiceConnector is used for figuring out the type of the Service Connectors and setting their parameters. This time the inheritance of the GenericServiceConnector will not be changed. Instead, it will be used by the Touchpoint Server to determine the type of the Service Connectors so that you can know their schema. Next, when the Touchpoint is being configured, the parameters will again be set to the GenericServerConnector, thus propagating them to its children.
- When using a **virtual** Touchpoint Type the GenericServiceConnector, at least for now, is not used, because so far only the Intermediary Touchpoint Type is part of this scheme, and it does not connect to third-party systems.
- *HTTPClientConnector* – the HTTP Client used for sending data to Destinations by the Initiator and Intermediary Touchpoints. It is used by the Sender script and made available to the Touchpoints through the `sendToDestinations()` method.

- *MemoryPropertiesConnector* – a Script Connector used by the MemoryProperties Store for holding the request-out URLs of the Destinations. It offers the ability to communicate with a running Touchpoint Instance and add/remove Destinations to it (for more details see the description of the MemoryProperties Store). This Connector mimics the behavior of the Properties Connector with the major difference that does not store the provided data in a file.

- **Properties:**

- *MemoryProperties* – a Properties Store used for communicating to a running Touchpoint AssemblyLine. It relies on the MemoryProperties Connector for storing the passed data in memory. A separate ConfigInstance is started for each Touchpoint, so each one has its own MemoryProperties Store and there is no risk of interweaving communication messages.

The Communication procedure is as follows:

1. The Touchpoint Server uses the Remote Server API to store a specific property – `com.ibm.di.tp.destinations`, in the MemoryProperties store. Its value is a `java.util.List` of `java.util.Maps` holding the provided destination parameters (for example, request-out and request-error URLs) configured for the Touchpoint.
2. Each time a Destination is added or removed from an Initiator or Intermediary Touchpoint Instance the Server updates this property's value, providing the current list of URLs.
3. On each iteration the Initiator or Intermediary Touchpoint AssemblyLine gets the current value of this property and send its data to the stored URLs (this is done by the `sendToDestinations()` routine).

The base Touchpoint Template provides a properly configured MemoryProperties Store. Every time a Touchpoint is created the Touchpoint Server checks if the MemoryProperties is missing and if so, adds it to the configuration. Thus, if you do not configure this Store in your custom Templates the default one will still be available. On the other hand, if you modify the MemoryProperties to alter the communication behavior (for example, use a Properties Connector, so that the Destination URLs are persisted to a file), the Touchpoint Server will not overwrite their new configuration.

- **Scripts:**

- *Sender* – a script providing the `sendToDestinations()` method. This routine reads the content of the `com.ibm.di.tp.destinations` property in the MemoryProperties store to get the Destination request-out/request-error URLs and sends the provided work entry using the `HttpClientConnector`.

Note: The request-error URL is not used by default; it is only provided to the TouchpointTemplate for you, so you can enhance the default error recovery mechanism or implement a custom one.

- *Utils* – this script provides a set of utility functions used by the Touchpoint AssemblyLines.

Custom templates let you provide complex/customized behavior in an AssemblyLine and expose it as a new Touchpoint Type. The default base template is used for direct provisioning of Tivoli Directory Integrator Components and an Intermediary Touchpoint without requiring you to know anything about what is happening under the hood of TDI. However, you may want to add more logic and/or Connectors to a Touchpoint Instance than a single Component. For this purpose the Touchpoint Server accepts custom templates to facilitate new custom Touchpoint Types and behavior.

A custom template's structure *should* be like the one of the base Template. However, if you do not need your custom Touchpoint Type to support all Touchpoint roles you may provide only a subset of the AssemblyLines. The minimum requirements for each role are:

- **Provider role.** To support it, the custom Template should contain the *ProviderHandler* AssemblyLine and the following Library Components: the *Utils* Script Component and the *GenericServiceConnector*.

Note: The base Template's ProviderServer AL is used to delegate request to all Touchpoint Instances. Thus, even if a ProviderServer AL exists in a custom template, it won't be used by the Touchpoint Server. The one of the base template will be used instead.

- **Initiator role.** This role requires the Initiator AssemblyLine and the following Library Components: the *Sender* and *Utils* Script Components, *MemoryPropertiesConnector*, *GenericServiceConnector* and *HttpClientConnector*.

Note: The MemoryProperties Store is not listed, because if missing, it will be added by the Touchpoint Server.

- **Intermediary role.** This role requires the *IntermediaryHandler* AssemblyLine and the following Library Components: the *Sender* and *Utils* Script Components, *MemoryPropertiesConnector* and *HttpClientConnector*.

If you try to create a Touchpoint Instance in a role which requirements are not fulfilled by the custom Type's Template an Exception will be thrown.

You must keep in mind these important points when editing the base Touchpoint Template or creating a custom one.

1. Their Service Connectors (the ones communicating with a third-party system) should inherit from the *GenericServiceConnector* located in the Library.
2. Only one Service Connector should be used in a Touchpoint AssemblyLine. If several are provided, they all will get identical configurations which will render them useless.
3. If you want to change the Store used for communicating Destination URLs, it still should be called *MemoryProperties*.

Resource Persistence

The Touchpoint Server supports persistence of the Atom documents, which could be either automatically generated by it or provided by a remote client. The persistence that is used by the server is stored in a configured folder, in a tree-like structure. The default persistence directory is `solution_directory/tp_state`.

The persisted resources are read back again when the Touchpoint Server is starting up. At this point the Touchpoint Server restores the complete resource tree. The Touchpoint Server persist the Touchpoint Instance configurations in order for a Touchpoint Instance to survive a restart of the server. A Touchpoint Instance is restarted when all of the following conditions are met:

- All of the required configurations for that Touchpoint Instance are available in the persistence storage.
- The remote Touchpoint Provider is up and running.
- There is no other Touchpoint Instance with that configuration running on the Touchpoint Provider.

In the case when the remote Touchpoint Provider is not running the Touchpoint Instance will not be automatically started right after the Touchpoint Provider comes back up. You must either make sure that the Touchpoint Provider is running prior to starting the Touchpoint Server or send a GET request to the Touchpoint Types Feed resource to force the Touchpoint Server to refresh the connection to the remote Touchpoint Provider.

Editing of the files in the persistence directory is not advisable. Currently, those files should be edited by the Touchpoint Server only.

Touchpoint Schema

Touchpoint Server communication protocol

This section specifies the communication protocol between the Touchpoint Server and a client that is using it for provisioning of Touchpoint Instances. This section does not describe the communication between a client and the Touchpoint Instance itself.

The Touchpoint Server is providing access to the various resources that are involved in the definition of a Touchpoint Instance. The representation of these resources is in a tree-like form. The access to each resource is done using Atom documents over the HTTP/HTTPS protocol.

This is the schema used by the Touchpoint Server:

* These tree nodes are available in certain cases. For details see the table below.

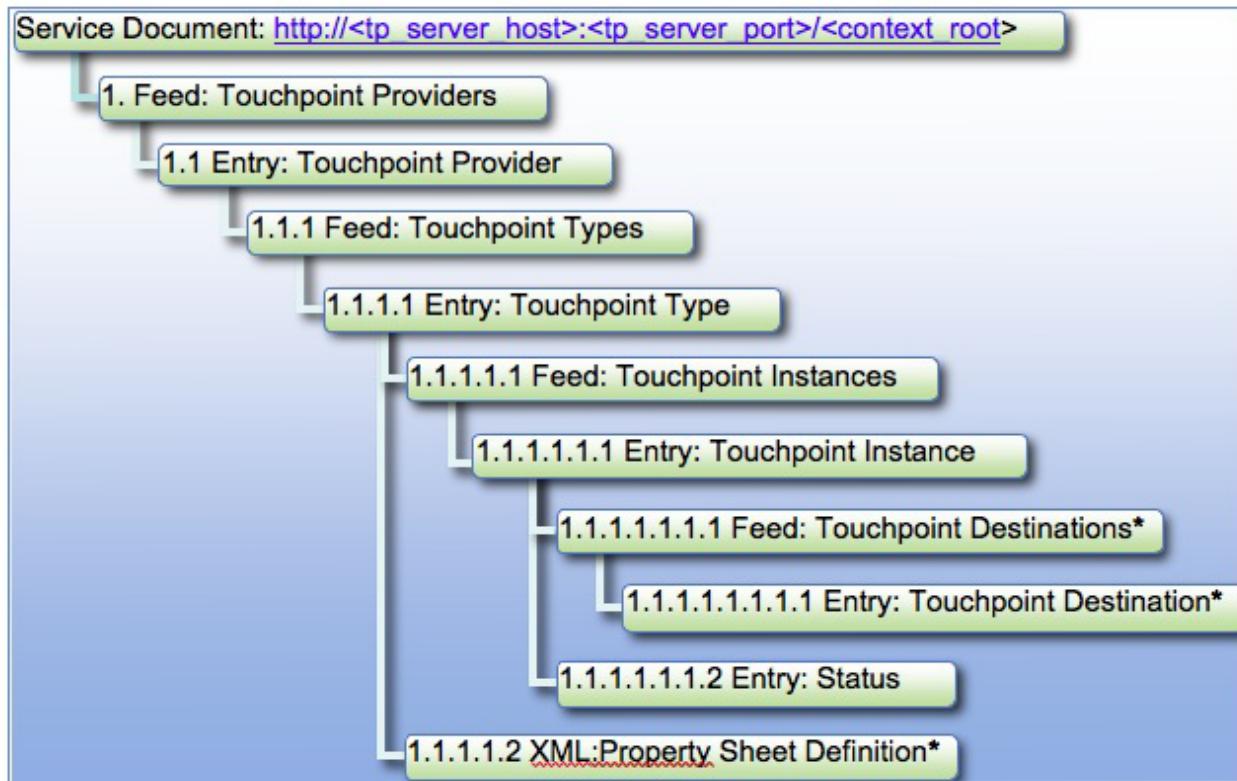


Figure 1. Touchpoint server schema tree

In the above schema the following variables are used:

- **tp_server_host** - this is the host address the Touchpoint Server is listening on
- **tp_server_port** - this is the port the Touchpoint Server is listening on (defaults to 1098)
- **context_root** - this is the context root under which the Touchpoint Server application is available (defaults to "tp")

The navigation of the tree is done in a ReSTful way, meaning that client applications should only know the entry point (that is, the URL of the Service Document) and the type of references (Atom links) the Touchpoint Server defines for accessing each of the nodes in the resource tree. These references (URLs) are automatically generated by the Touchpoint Server. Once obtained, the URLs will stay the same until the Touchpoint Server is updated to a newer version. This implies that client applications can "remember" the obtained URLs between updates of the Touchpoint Server, but should work their way back to the particular resource URL if the Touchpoint Server is updated.

According to the protocol specified in the table below, the following steps should be performed in order for a client application to navigate to the Touchpoint Instance Feed starting from the Service Document.

1. Send an HTTP GET request to the Service Document URL. This will return the Service Document from which the Touchpoint Providers Feed URL could be obtained.
2. Send an HTTP GET request to the Touchpoint Providers Feed URL. This will return the Feed Document from which the Touchpoint Provider Entry Reference URL could be obtained.
3. Send an HTTP GET Touchpoint Provider Entry Reference URL. This will return the Entry Document representing the particular Touchpoint Provider. This Entry contains the URL to the Touchpoint Type Feed.
4. Send an HTTP GET request to the Touchpoint Type Feed URL. This will return the Feed Document containing complete copies of all the Touchpoint Type Entries valid for this context. From a Touchpoint Type Entry the client application can obtain the Touchpoint Instance Feed URL.

The table below describes the allowed operations for each resource. Additionally the following points are applicable for the whole resource tree:

- Each entry has a link with relation "self" that points to the standalone Entry Document.
- Resources that accept HTTP methods PUT and DELETE have a link with relation "edit". All such request should be sent to that link (URL).
- Every request to the Touchpoint Server is annotated with the HTTP ETag response-header as defined by the HTTP specification. The ETag value can be used in conjunction with the request-headers If-Match, If-None-Match and If-Range to let the Touchpoint Server know what the client application's preconditions are before servicing the request.

Table 31. Allowed operations per resource

Resource	GET	POST	PUT	DELETE
Service Document	Retrieves the Service Document which contains a list of the available services. The URL to the Touchpoint Providers Feed is set as "href" attribute to a collection that belongs to the category "connectivity-provider".	N/A	N/A	N/A
1. Feed: Touchpoint Providers Categories (term: scheme): connectivity-provider: http://www.ibm.com/xmlns/prod/scmp#resource	Retrieves the list of Touchpoint Provider Entries. All entries are references to the actual Entry Documents representing the available Touchpoint Providers.	N/A	N/A	N/A
1.1 Entry: Touchpoint Provider	Retrieves a Touchpoint Provider Entry the way it was configured in the Touchpoint Server configuration file. This Entry provides some additional details in the <{http://www.ibm.com/xmlns/prod/scmp}:data/> element. The link to the Touchpoint Types Feed is provided using a link with relation "http://www.ibm.com/xmlns/prod/scmp#touchpoint"	N/A	N/A	N/A

Table 31. Allowed operations per resource (continued)

Resource	GET	POST	PUT	DELETE
1.1.1 Feed: Touchpoint Types Categories (term: scheme): touchpoint: http://www.ibm.com/xmlns/prod/scmp#resource	Retrieves the list of Touchpoint Type Entries. These entries are complete copies of the actual Entry Documents representing the Touchpoint Types.	N/A	N/A	N/A
1.1.1.1 Entry: Touchpoint Type Categories (term: scheme): touchpoint: http://www.ibm.com/xmlns/prod/scmp#resource resource-type: http://www.ibm.com/xmlns/prod/scmp#aspect A term uniquely identifying a Touchpoint Type: http://www.ibm.com/xmlns/prod/scmp#touchpoint-type	Retrieves a Touchpoint Type Entry. The URL to the Touchpoint Instance Feed is provided as a link with relation "http://www.ibm.com/xmlns/prod/scmp#instance-feed" The URL to the Property Sheet Definition XML is provided as a link with relation "http://www.ibm.com/xmlns/prod/scmp#property-sheet-definition". Note: The virtual Touchpoint Type does not have a Property Sheet Definition, because no Connectors need to be configured for Intermediary Touchpoints.	N/A	N/A	N/A
1.1.1.1 Feed: Touchpoint Instances Categories (term: scheme): touchpoint: http://www.ibm.com/xmlns/prod/scmp#resource	Retrieves the list of Touchpoint Instance Entries. All entries are references to the actual Entry Documents representing the available Touchpoint Instances.	Creates a new Touchpoint Instance Entry. The entry MUST contain a data element that contains the Touchpoint Instance's configuration. The entry must contain a category from the "http://www.ibm.com/xmlns/prod/scmp#touchpoint-role" scheme.	N/A	N/A

Table 31. Allowed operations per resource (continued)

Resource	GET	POST	PUT	DELETE
1.1.1.1.1 Entry: Touchpoint Instance Categories (term: scheme): touchpoint: http://www.ibm.com/xmlns/prod/scmp#resource provider-tp OR initiator-tp OR intermediary-tp: http://www.ibm.com/xmlns/prod/scmp#aspect A term uniquely identifying a Touchpoint Type: http://www.ibm.com/xmlns/prod/scmp#touchpoint-type	<p>Retrieves the Touchpoint Instance Entry. This Entry contains three links to the resources that describe the Touchpoint Instance:</p> <ul style="list-style-type: none"> An URL used for updating/deleting this Touchpoint Instance is provided using a link with relation "edit"; An URL to the Destination Feed is provided using a link with relation http://www.ibm.com/xmlns/prod/scmp#tp-destination Note: The Provider Touchpoints do not support Destinations, so for them this link is missing. An URL to the Status Entry is provided using a link with relation http://www.ibm.com/xmlns/prod/scmp#status. An URL to the Touchpoint Type Entry to which this Touchpoint Instance belongs. The link has relation http://www.ibm.com/xmlns/prod/scmp#resource-type 	N/A	<p>Updates the Touchpoint Instance entry. The provided entry must contain the complete copy of the Entry Document and not just the changes. This operation cannot change the role of the Touchpoint Instance. This operation restarts a running Touchpoint Instance in order to reconfigure it.</p>	<p>Deletes the Touchpoint Instance Entry.</p>
1.1.1.1.1 Feed: Touchpoint Destinations	<p>Retrieves the Touchpoint Instance Destinations Feed. This Feed could contain multiple Touchpoint Instance Destination Entries.</p>	<p>Creates a new Touchpoint Instance Destination Entry. It MUST contain a data element configuring the request-out URL to the Destination.</p>	N/A	N/A

Table 31. Allowed operations per resource (continued)

Resource	GET	POST	PUT	DELETE
1.1.1.1.1.1.1 Entry: Touchpoint Destination Categories (term: scheme): tp-destination: http://www.ibm.com/xmlns/prod/scmp#resource	Retrieves the Touchpoint Destination Entry that contains the request-out URL to the remote HTTP Service in the <{http://www.ibm.com/xmlns/prod/scmp}:data/> element. This entry provides a link with relation "edit" to enable updates/deletes on this resource.	N/A	Updates the Touchpoint Instance entry. The provided entry must contain the complete copy of the Entry Document and not just the changes. This operation cannot change the role of the Touchpoint Instance. This operation does NOT restart a running Touchpoint Instance in order to alter the request-out URL.	Deletes the Touchpoint Destination Entry.
1.1.1.1.1.2 Entry: Status Categories (term: scheme): touchpoint: http://www.ibm.com/xmlns/prod/scmp#resource status: http://www.ibm.com/xmlns/prod/scmp#aspect	Retrieves the Touchpoint Instance Status Entry which describes the operational state of the particular Touchpoint Instance. It is contained inside the <{http://www.ibm.com/xmlns/prod/scmp}:data/> element. Note: For Provider and Intermediary Touchpoints the status also contains one request-in URL, which should be used by clients when querying the Touchpoint.	N/A	N/A	N/A
1.1.1.2 XML: Property Sheet Definition	Retrieves the Property Sheet Definition for the selected Touchpoint Type. It holds the schema of a TDI Connector in the form of an XML document.	N/A	N/A	N/A

Touchpoint Configuration

This section describes the schema of the configuration data that one must provide in order to configure and start a Touchpoint Instance.

Every configuration data element is placed inside a **<data>** element within the Atom document. The namespace this data element must belongs to is: <http://www.ibm.com/xmlns/prod/scmp>

Instance Configuration

This configuration data specifies:

- Role that the Touchpoint Instance will be running in. The Touchpoint Instance relies on this data to decide which AL from the template to use. In the atom document below, it is denoted by the "{role}" token.in the category element. Its supported values are provider-tp, initiator-tp and intermediary-tp.
- Admin state of the created Touchpoint Instance. It is marked by the "{admin_state}" token in the Atom document. The supported values are enabled and disabled.
- Property Sheet XML containing the configuration parameters for Touchpoint Instance's Service Connector. The {param_name} is determined by the Property Sheet Definition of the chosen Touchpoint Type for standard Types or the Property Sheet Definition of the Service Connector for custom Types. The "{param_value}" is determined by the user depending on the wanted configuration. Besides configuration parameters, you can also set the mode of the Service Connector by setting a parameter with {param_name} equal to "\$initMode" and a string value with the mode name (for example, Iterator, AddOnly). For details about this parameter see section "Property sheet definitions" on page 281.
- The two parameters {TouchpointID} and {version} are provided in order to allow you to specify additional information for the particular Touchpoint Instance, which has meaning for the creator only. The creator is responsible for ensuring these values are valid in the context of the client application. These values are not interpreted by the Touchpoint Server in any way, it only persists them.

The POSTed Atom Document when creating a Touchpoint Instance Entry Resource is:

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <id>{id}</id>
  <title>Touchpoint Instance Title</title>
  <author><name>Author Name</name></author>
  <content/>
  <category term="{role}" scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-role" />
  <scmp:data
    xmlns:scmp="http://www.ibm.com/xmlns/prod/scmp" xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scmp
    http://localhost:1098/tp/schema/touchpoint.xsd" >
    <scmp:touchpoint>
      <scmp:admin-state>{admin_state}</scmp:admin-state>
      <touchpointID>{touchpoint_id}</touchpointID>
      <version>{version}</version>
      <scmp:propertySheet>
        <scmp:property propertyName="{param_name}">
          <scmp:value>{param_value}</scmp:value>
        </scmp:property>
        ...
      </scmp:propertySheet>
    </scmp:touchpoint>
  </scmp:data>
</entry>
```

Please note that an ID for the created Touchpoint Instance can also be provided in place of the "{id}" token. However, no matter what the passed value is, the Touchpoint Server will overwrite it with an automatically generated one. This way it guarantees the uniqueness of the Touchpoint Instance ID.

Destination Configuration

Both the Intermediary and Initiator Touchpoints require a Destination to be configured, before they become operational. Furthermore, they support multiple such Destinations and the ability to add and remove them at runtime.

This is done by POSTing an Atom Document on the Touchpoint Destination Feed URL of the created Touchpoint Instance. Keep in mind that no such URL is available for Provider Touchpoints. Here is how it should look like:

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <scmp:data
    xmlns:scmp="http://www.ibm.com/xmlns/prod/scmp" xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scmp
    http://localhost:1098/tp/schema/touchpoint.xsd" >
```

```

<scmp:destination>
  <scmp:request-out>{request-out_URL}</scmp:request-out>
  <scmp:request-error>{request-error_URL}</scmp:request-error>
</scmp:destination>
</scmp:data>
</entry>

```

As can be seen from this snippet, only the "{request-out_URL}" is needed for configuring a Destination; the "{request-error_URL}" is optional.

Touchpoint Instance communication protocol

This section describes the protocol used to communicate with a Touchpoint Instance. In most cases a Touchpoint Instance will be derived from the Touchpoint Template.

Provider Touchpoint

The Provider Touchpoint handles HTTP methods described in the following table:

Table 32. Provider Touchpoint HTTP methods

HTTP Method	URL Query Parameters	Connector mode	HTTP Request Content	HTTP Response Content	HTTP Response Code
GET	-	Iterator	-	all Entries found	"200 OK" if at least one Entry was found "404 Not Found" if no Entries are found
GET	link criteria	Lookup	-	all Entries found	"200 OK" if at least one Entry was found "404 Not Found" if no Entries are found
POST	-	AddOnly	Entry to be added	-	"201 Created" if the operation was successful
PUT*	link criteria	Update	Entry with updated attributes	-	"201 Created" if the Entry did not exist and was added "204 No Content" if a single Entry matches the link criteria and the Entry was updated successfully
DELETE	link criteria	Delete	-	-	"204 No Content" if the operation was successful (the operation will fail if multiple Entries match the link criteria)

(*) Note that there is a difference in the handling of the PUT method between our implementation and the HTTP 1.1 specification (<http://tools.ietf.org/html/rfc2616#section-9.6>). According to the HTTP specification, a PUT request is supposed to replace the whole resource. In our implementation if the entry exists we do not replace it as a whole, but only replace the specified attributes.

The Link Criteria for the Connector operations is derived from the query parameters of the requested URL. For example, a GET request with URL "http://localhost/mytp?username=jsmith" will result in a Lookup operation with link criteria "username=jsmith". Each query parameter corresponds to an EXACT match criterion. The criteria derived from multiple query parameters are combined using the 'AND' logic operator. For example: "?firstname=john&age=50" corresponds to ((firstname equals "john") AND (age equals "50")).

Query parameters are required for PUT and DELETE requests. POST requests are not expected to contain query parameters. If a GET request contains query parameters, it is translated to a Lookup mode operation. Otherwise it is translated to an Iterator mode operation.

On GET requests you can use an optional HTTP header named "X-TDI-TP-SizeLimit" to limit the number of returned Entries. The value of the header must be an integer larger than zero.

All HTTP Methods interpreted by the default Touchpoint Template are compliant with the HTTP Specification according their safety and idempotence characteristics.

Initiator Touchpoint

The Initiator Touchpoint Instance acts as an HTTP client. It has an Iterator Connector which produces Entry objects that the AssemblyLine sends to a configured Destination URLs. For each Entry it sends a single POST request, whose content is an XML representation of the Entry.

Intermediary Touchpoint

The Intermediary Touchpoint Instance is similar to both the Provider and Initiator roles. It accepts requests on a particular request-in URL as a Provider and sends the received data to multiple Destinations as an Initiator. Due to this forwarding functionality it can be used as moderator between several Touchpoint Instances from the other roles.

Representation of Entry objects as HTTP content

Example:

```
<tp:data xmlns:tp="http://www.ibm.com/xmlns/prod/tdi/711/tp">
  <tp:entry>
    <tp:attribute name="username">
      <tp:value><![CDATA[jsmith]]</tp:value>
    </tp:attribute>

    <tp:attribute name="mail">
      <tp:value>jsmith@ibm.us.com</tp:value>
      <tp:value>john.smith@gmail.com</tp:value>
    </tp:attribute>
  </tp:entry>
</tp:data>
```

The HTTP content must be encoded in "UTF-8". It must contain a single data element which can contain zero or more entry elements.

XML Schema description on the touchpoint data format:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace="http://www.ibm.com/xmlns/prod/tdi/711/tp"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.ibm.com/xmlns/prod/tdi/711/tp" >
```

```

<element name="data" type="tns:TouchpointDataType" />
<element name="entry" type="tns:EntryType" />
<element name="attribute" type="tns:AttributeType" />
<element name="property" type="tns:PropertyType" />
<element name="value" type="string" />
<complexType name="TouchpointDataType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="tns:entry" />
  </choice>
</complexType>
<complexType name="EntryType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="tns:property" />
    <element ref="tns:attribute" />
  </choice>
</complexType>
<complexType name="AttributeType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="tns:property" />
    <element ref="tns:attribute" />
    <element ref="tns:value" />
  </choice>
  <attribute name="name" type="string" use="required" />
  <attribute name="namespaceURI" type="anyURI" />
</complexType>
<complexType name=".PropertyType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required" />
      <attribute name="namespaceURI" type="anyURI" />
    </extension>
  </simpleContent>
</complexType>
</schema>

```

Touchpoint Status Entry schema

The status of the Touchpoint Instance can be retrieved by sending an HTTP GET request to the URL of the Status Entry as described by section “Touchpoint Server communication protocol” on page 272.

```

<entry xmlns="http://www.w3.org/2005/Atom">
  <id>Status ID</id>
  <link href="{touchpoint_instance_status_URL}" type="application/atom+xml;type=entry" rel="self"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#resource" term="touchpoint"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#aspect" term="status"/>
  <scmp:scmp
    xmlns:scmp="http://www.ibm.com/xmlns/prod/scmp"
    xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scmp
    http://localhost:1098/tp/schema/touchpoint.xsd" >
    <scmp:touchpoint-status>
      <scmp:request-in>{request-in_URL}</scmp:request-in>
      <scmp:op-state>{op-state}</scmp:op-state>
    </scmp:touchpoint-status>
  </scmp:scmp>
</entry>

```

The returned Atom Entry Document contains a `<data>` element that describes the status of the Touchpoint Instance. The data element belongs to the `http://www.ibm.com/xmlns/prod/scmp` namespace. It has the following syntax:

- `{op-state}` – is the string keyword describing the current state of the Touchpoint Instance as defined in section “Touchpoint Instance” on page 265.
- `{request-in_URL}` – is the URL for accessing Provider and Intermediary Touchpoint Instances.

Property sheet definitions

The Property Sheet Definition is an XML document that determines the schema of a Tivoli Directory Integrator Connector. It can be obtained from a link inside the Touchpoint Type Entry (as described in section “Touchpoint Server communication protocol” on page 272).

Depending on the Touchpoint Type the Property Sheet Definition varies:

- For standard Types, it contains the schema of the TDI Connector corresponding to the Touchpoint Type.
- For custom Types, it contains the schema of the Service Connector in the custom Touchpoint Template.
- For the virtual Type (`virtual://Intermediary`) there is no Property Sheet Definition, as there is no third-party system to communicate to (this role relies solely on HTTP).

Besides the schema parameters of a TDI Connector the Property Sheet Definition contains the modes, supported by the Connector. They are stored as optional values for the `propertyDefinition` with name `$initMode`. Their values directly match the Connector mode names – “Iterator”, “AddOnly”, “CallReply”, and so forth.

Here is an example Property Sheet Definition for the File Connector (Touchpoint Type `system:/Connectors/ibmdi.LDAP`):

```
<?xml version="1.0" encoding="UTF-8"?>
<propertySheetDefinition xmlns="http://www.ibm.com/xmlns/prod/scmp">
    <propertyDefinition required="true" hidden="false" readonly="false"
        propertyType="string" multiple="false"
        propertyName="ldapUrl">
        <label label="LDAP URL" lang="en"/>
        <!--one label for the different languages supported by TDI -->
    </propertyDefinition>
    <!--the rest of LDAP Connector's parameters -->
    <propertyDefinition required="false" readonly="false" propertyType="string"
        multiple="false" propertyName="$initMode">
        <label label="$initMode" lang="en"/>
        <!--one label for the different languages supported by TDI -->
        <option>
            <value>AddOnly</value>
            <label label="AddOnly" lang="en"/>
            <!--one label for the different languages supported by TDI -->
        </option>
        <option>
            <value>Iterator</value>
            <label label="Iterator" lang="en"/>
            <!--one label for the different languages supported by TDI -->
        </option>
        <!--the rest of modes supported by the LDAP Connector -->
    </propertyDefinition>
</propertySheetDefinition>
```

To shorten the listing, only one of its configuration parameters and the `$initMode` property definition are included. As can be seen this Connector has a parameter named `ldapUrl` which is required and its value is a string. Also, its English label as can be seen in the CE is **LDAP URL** (the labels for the rest of the

languages supported by Tivoli Directory Integrator are skipped). The \$initMode parameter is also present, and as can be seen from its optional values this Connector supports both Iterator and AddOnly modes (and several others which are skipped).

Property Sheet Definitions significantly assist you when creating Touchpoint Instances, as you do not need to know the schema of the Connector in advance. Relying on their information, you can perform this task, much like you configure Connectors inside TDI's Configuration Editor – by seeing the needed parameters, checking their expected values (a string, a number or a set of predefined values) and providing them in the configuration.

XML Schema locations

An XML Schema document is provided for each scmp:data element. The document defines all elements which appear inside the scmp:data element.

The location of the schema document is specified in an xsi:schemaLocation (XML Schema Instance location) attribute defined on the scmp:data element. For example:

```
<scmp:data
  xmlns:scmp="http://www.ibm.com/xmlns/prod/scmp"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scmp
  http://localhost:1098/tp/schema/tdi-connectivity-provider.xsd">

<scmp:connectivity-provider>
```

The location of the schema is a valid URL which can be de-referenced by web clients: Any client of the Touchpoint Server can resolve the URL which appears inside the xsi:schemaLocation attribute and access the actual schema document.

Moreover if the schema contains a reference to another schema document (for example, via "import", "include" or "redefine" XML Schema element), the URL in the reference can be resolved by web clients to obtain the referred schema.

Error flows

Appropriate error messages are issued to the standard log if an error is thrown. The possible situations for which an error might be produced are:

- Incorrect configuration of Touchpoint Server.
- Exceptions thrown as result of communication problems with the Persistence Store (File System errors).
- Error in communication with the clients of the Touchpoint Server.
- Error in communication with of the Touchpoint Server with Tivoli Directory Integrator.

When the error is due to an invalid information/request sent by the user, and which violates the Touchpoint Server protocol, it will produce an XML document with the following syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:error xmlns:ns2="http://www.ibm.com/xmlns/prod/scmp">
  <creation-time>2010-02-23T14:49:06.384+02:00</creation-time>
  <code>100005</code>
  <details>
    <detail>
      <name>schema</name>
      <value>http://www.ibm.com/xmlns/prod/scmp#touchpoint-role</value>
    </detail>
  </details>
  <native-msgid>ABCD1234E</native-msgid>
  <summary>Missing role category</summary>
</ns2:error>
```

Where:

- **creation-time** element denotes the time the error has occurred in a format specified by <http://www.w3.org/TR/1998/NOTE-datetime-19980827>.
- **code** element is one of the following codes:
 - 100000 – unknown error occurred (could not narrow it further)
 - 100001 – Missing required atom:link. Details element will provide: **rel** - relationship name of expected link
 - 100002 – Missing required scmp:data element –Details element will provide: **qname** - missing element qname
 - 100003 – Invalid atom:entry in POST/PUT operation (for example, parse error)
 - 100004 – Invalid value for scmp:data element – Details element will provide: **qname** - invalid element qname; **value** - invalid supplied value
 - 100005 – Missing required atom:category. Details element will provide: **scheme** - expected scheme name
 - 100006 – Invalid atom:category value. Details element will provide: **scheme** - expected scheme name; **term** - invalid supplied term
 - 100007 – Too many atom:link for given relationship. Details element will provide: **rel** - relationship name of extra link
 - 100008 – Too many atom:category values for scheme. Details element will provide: **scheme** - overpopulated scheme name
 - 330000 – Default connectivity-provider-specific error (unable to narrow down further)
- **details** element is containing more details for the specific error. See the particular error code to find out what kind of detailed information is expected for each error.
- **native-msgid** element denotes the message short id.
- **summary** element contains the human readable error message.

When the error is not caused by protocol violation but is from a different source, a human readable representation is returned in a plain text format. The error also contains the exception stack trace so you can report it to the Touchpoint Server administrator in order for it to be resolved.

Configuration

The Touchpoint Server runs inside a web container. The default web container shipped with Tivoli Directory Integrator is configured by the following properties in `global.properties` or `solution.properties`:

- `tp.server.on` – specifies whether the bundled web container and the Touchpoint Server should be started. Default value: `false`.
- `tp.server.port` – specifies the port the web container will be listening on. Default value: 1098.
- `tp.server.auth` – specifies whether the Touchpoint Server will use HTTP Basic authentication. Default value: `false`.
- `tp.server.auth.realm` – specifies the realm HTTP basic authentication. Default value: "Tivoli Directory Integrator Touchpoint Server".

The Touchpoint Server first considers the value of the `api.remote.bind.address` property and if that is not set, the value of the `com.ibm.di.default.bind.address` property. In this way it is able to effectively filter the access to "multi-homed" hosts.

The web container is able to use SSL for securing the transportation layer. It reuses the Remote API's settings and is enabled by setting the `api.remote.ssl` property. SSL client authentication is enabled by the `api.remote.ssl.client.auth.on` property. The server SSL keys are configured using the well known Remote API properties:

- `api.keystore`

- api.client.keystore.pass
- api.client.key.pass
- api.client.keystore.type

HTTP basic authentication (<http://tools.ietf.org/html/rfc2617>) can be configured using the tp.server.auth and tp.server.auth.realm properties. It is disabled by default. See section "Authentication" on page 285 for more information on authentication.

The configuration of the Touchpoint Server is specified using an XML file. The path to this file is specified in the global.properties or solution.properties file using the property tp.server.config. An example Touchpoint Server configuration file is shipped in the etc directory of the Tivoli Directory Integrator installation.

The following syntax is used by the Touchpoint Server configuration file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tp:tpServerConfig xmlns:tp="http://www.ibm.com/xmlns/prod/tdi/711/tp" tp:version="1.0">

    <!-- specifies the encryption settings used when encrypting passwords -->
    <tp:encryptionConfig stash="idisrv.sth">
        <tp:keyStore>testserver.jks</tp:keyStore>
        <tp:keyStoreType>jks</tp:keyStoreType>
        <tp:keyAlias>server</tp:keyAlias>
        <tp:transformation>RSA</tp:transformation>
    </tp:encryptionConfig>

    <tp:templateConfig>
        <tp:baseTemplate>etc/TouchpointTemplate.xml</tp:baseTemplate>

        <!-- Specify the path to the directory that holds the Touchpoint templates. -->
        <!--
        <tp:customTemplatesDir>templates</tp:customTemplatesDir>
        -->
    </tp:templateConfig>

    <!-- specifies the persistence settings that configure the place to persist the state -->
    <tp:persistenceConfig>
        <tp:enabled>true</tp:enabled>
        <tp:location>tp_state</tp:location>
    </tp:persistenceConfig>

    <!-- configures the touchpuoint providers (nodes) -->
    <tp:nodeConfigs>
        <!-- Default connection to the local server -->
        <tp:tdiNodeConfig tp:local="true" tp:id="default">
            <!-- The host of the remote node which all
                Provider Touchpoint Instances will receive requests on -->
            <tp:providerHost>localhost</tp:providerHost>
            <!-- The port of the remote node which all
                Provider Touchpoint Instances will receive requests on -->
            <tp:providerPort>1097</tp:providerPort>

            <tp:title>Example Touchpoint Provider</tp:title>
            <tp:author>John Doe</tp:author>
            <tp:email>jdoe@example.org</tp:email>
            <tp:summary>Example Touchpoinet Provider Atom Entry</tp:summary>

            <tp:conatct>Local Administrator</tp:conatct>
            <tp:location>Main building, 5th fl.</tp:location>
            <tp:organization>Example Organization</tp:organization>
        </tp:tdiNodeConfig>

        <!-- Here is an example of a remote server connection -->
        <!--
            <tp:tdiNodeConfig id="remote" local="false">

```

```

<tp:title>Example Touchpoint Provider</tp:title>
<tp:author>John Doe</tp:author>
<tp:email>jdoe@example.org</tp:email>
<tp:summary>Example Touchpoint Provider</tp:summary>

<tp:host>localhost</tp:host>
<tp:port>1099</tp:port>
<tp:user>username</tp:user>
<tp:password protect="true" encrypted="false">password</tp:password>

<tp:contact>Jack Smith</tp:contact>
<tp:location>5th fl.</tp:location>
<tp:organization>Example Organization</tp:organization>

<tp:providerHost>localhost</tp:providerHost>
<tp:providerPort>1097</tp:providerPort>
  </tp:tdiNodeConfig>
-->
</tp:nodeConfigs>
</tp:tpServerConfig>

```

Authentication

There are two aspects of the Touchpoint Server related to authentication:

- being an HTTP server
- being a client of the remote RMI Server API of the Tivoli Directory Integrator Servers, which are configured as connectivity providers.

As an HTTP server, the Touchpoint Server supports HTTP basic authentication of HTTP clients. It does not use a separate user registry. Instead the Touchpoint Server delegates authentication requests to the Server API of the local Tivoli Directory Integrator Server (the one that hosts the Touchpoint Server).

As a remote Server API client, the Touchpoint Server needs to authenticate against the remote Tivoli Directory Integrator Server like any other Server API client would do. Note that authentication is not required when the connectivity provider is the local Tivoli Directory Integrator Server.

For more on Server API authentication, see the appendix called *Server API in IBM Tivoli Directory Integrator V7.1.1 Reference Guide*.

Examples

Shipped example

The main example demonstrating the use of the Touchpoint Server functionality in Tivoli Directory Integrator, demonstrating example steps for creating both Provider and Initiator Touchpoint Instances, is shipped with the installation. The steps are spelled out in the document found at *TDI_Install_dir/examples/TouchpointClient/Touchpoint_Example.pdf*. In addition to that, a sample implementation of those steps is provided to demonstrate how they might look when written in the Java programming language. The Java code is structured in two packages, as follows:

- **com.ibm.di.tp.client.api** – this package contains the code demonstrating how the communication with the Touchpoint Server is performed. It contains nothing but methods used for querying the Touchpoint Server.

Note: This code depends on Apache HttpClient v3.x.

- **com.ibm.di.tp.client.gui** – this package contains a sample UI client that uses the *com.ibm.di.tp.client.api* package to interact with the Touchpoint Server in order to provide Touchpoint Instances. You can use this UI for testing purposes when provisioning Touchpoint Instances. You may start this utility using the provided scripts: *startClient.bat* and *startClient.sh*.

Example steps for creating a Touchpoint Instance using a JDBC Connector

In order to provision a Touchpoint Instance, you need to send an HTTP POST request to the Touchpoint Server. The URLs for the requests are obtainable according to the application schema. For the purpose of this example we are using pseudo URLs as: <Resource Name URL>. You can obtain the appropriate URL at runtime.

Provider Touchpoint Instance

Create a Touchpoint Instance Entry resource:

POST <Touchpoint Instance Feed URL>

```
Body:<entry xmlns="http://www.w3.org/2005/Atom">
<id>some ID</id>
<title>Provider Touchpoint Instance</title>
<author><name>author_name</name></author>
<content/>
<category term="provider-tp" scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-role" />
<scmp:admin-state enabled="true" />
<scmp:touchpoint>
  <scmp:propertySheet>
    <scmp:property propertyName="jdbcSource">
      <scmp:value>some source value</scmp:value>
    </scmp:property>
    <scmp:property propertyName="jdbcDriver">
      <scmp:value>the driver class</scmp:value>
    </scmp:property>
    <scmp:property propertyName="jdbcTable">
      <scmp:value>the table name</scmp:value>
    </scmp:property>
    <!--The rest of the parameters required by a JDBC Connector-->
  </scmp:propertySheet>
  <scmp:admin-state enabled="true" />
</scmp:touchpoint>
</scmp:admin-state>
</entry>
```

The Touchpoint Server will return a response similar to:

```
201 Created
Location: <Touchpoint Instance Entry URL>
Body:
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="http://a9.com/-/spec/opensearch/1.1/"
       xmlns:ns3="http://www.w3.org/1999/xhtml">
  <id>generated_id</id>
  <updated>2010-02-17T18:33:55.302+02:00</updated>
  <title>Provider Touchpoint Instance</title>
  <link href="<Touchpoint Instance Entry URL>" type="application/atom+xml;type=entry" rel="self"/>
  <link href="<Touchpoint Instance Entry URL>" type="application/atom+xml;type=entry" rel="edit"/>
  <link href="<Touchpoint Type Entry URL>" type="application/atom+xml;type=entry" rel="type"/>
  <link href="<Touchpoint Instance Status Entry URL>" type="application/atom+xml;type=entry" rel="http://www.ibm.com/xmlns/prod/scmp#status-type"/>
  <author><name>author_name</name></author>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-type" term="system:/Connectors/ibmdi.JDBC"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-role" term="provider-tp"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#resource" term="touchpoint"/>
  <ns2:admin-state ns2="http://www.ibm.com/xmlns/prod/scmp">
    <ns2:touchpoint>
      <ns2:admin-state>enabled</ns2:admin-state>
      <ns2:propertySheet>
        <ns2:property propertyName="jdbcSource" xmlns="" xmlns:ns5="http://www.w3.org/2005/Atom">
          <ns2:value>some source value</ns2:value>
        </ns2:property>
        <ns2:property propertyName="jdbcDriver" xmlns="" xmlns:ns5="http://www.w3.org/2005/Atom">

```

```

<ns2:value>the driver class</ns2:value>
</ns2:property>
<!--The rest of the parameters required by a JDBC Connector-->
</ns2:propertySheet>
</ns2:touchpoint>
</ns2:data>

```

Note that the Touchpoint Server changes the ID of the entry to guarantee its uniqueness.

The URL used for accessing the created Touchpoint Instance can be retrieved through the Status Entry URL. To do this, send an HTTP GET request to URL <Touchpoint Instance Status Entry URL>. The received response looks like this:

```

200 OK
Location: <Touchpoint Instance Status Entry URL>
Body:
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="http://a9.com/-/spec/opensearch/1.1/"
       xmlns:ns3="http://www.w3.org/1999/xhtml">
  <id>generated_id</id>
  <link href="<Touchpoint Instance Status Entry URL>" type="application/atom+xml;type=entry" rel="self"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#resource" term="touchpoint"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#aspect" term="status"/>
  <ns2:data xmlns:ns2="http://www.ibm.com/xmlns/prod/scmp">
    <ns2:touchpoint-status>
      <ns2:request-in>Touchpoint Provider Request-in URL</ns2:request-in>
      <ns2:op-state>available</ns2:op-state>
    </ns2:touchpoint-status>
  </ns2:data>
</entry>

```

Initiator Touchpoint Instance

Create a Touchpoint Instance Entry resource:

POST <Touchpoint Instance Feed URL>

```

Body:
<entry xmlns="http://www.w3.org/2005/Atom">
  <id>some ID</id>
  <title>Initiator Touchpoint Instance</title>
  <author><name>author_name</name></author>
  <content/>
  <category term="initiator-tp" scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-role" />
  <scmp:touchpoint>
    <scmp:propertySheet>
      <scmp:property propertyName="jdbcSource">
        <scmp:value>some source value</scmp:value>
      </scmp:property>
      <scmp:property propertyName="jdbcDriver">
        <scmp:value>the driver class</scmp:value>
      </scmp:property>
      <scmp:property propertyName="jdbcTable">
        <scmp:value>the table name</scmp:value>
      </scmp:property>
      <!--The rest of the parameters required by a JDBC Connector-->
    </scmp:propertySheet>
    <scmp:admin-state>enabled</scmp:admin-state>
  </scmp:touchpoint>
</scmp:touchpoint>
</entry>

```

The Touchpoint Server returns a response similar to:

```

201 Created
Location: <Touchpoint Instance Entry URL>
Body:

```

```

<entry xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="http://a9.com/-/spec/opensearch/1.1/"
       xmlns:ns3="http://www.w3.org/1999/xhtml">
  <id>generated_ID</id>
  <updated>2010-02-17T18:33:55.302+02:00</updated>
  <title>Initiator Touchpoint Instance</title>
  <link href=<Touchpoint Instance Entry URL> type="application/atom+xml;type=entry" rel="self"/>
  <link href=<Touchpoint Instance Entry URL> type="application/atom+xml;type=entry" rel="edit"/>
  <link href=<Touchpoint Type Entry URL>" type="application/atom+xml;type=entry" rel="http://www.ibm.com/xmlns/prod/scmp#resource-type"/>
  <link href=<Touchpoint Instance Status Entry URL>" type="application/atom+xml;type=entry" rel="http://www.ibm.com/xmlns/prod/scmp#status"/>
  <link href=<Touchpoint Instance Destination Feed URL>" type="application/atom+xml;type=feed" rel="http://www.ibm.com/xmlns/prod/scmp#tp-destination"/>
  <author><name>author_name</name></author>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-type" term="system:/Connectors/ibmdi.JDBC"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-role" term="initiator-tp"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#resource" term="touchpoint"/>
  <ns2:data xmlns:ns2="http://www.ibm.com/xmlns/prod/scmp">
    <ns2:touchpoint>
      <ns2:admin-state>enabled</ns2:admin-state>
      <ns2:propertySheet>
        <ns2:property propertyName="jdbcSource" xmlns="" xmlns:ns5="http://www.w3.org/2005/Atom">
          <ns2:value>some source value</ns2:value>
        </ns2:property>
        <ns2:property propertyName="jdbcDriver" xmlns="" xmlns:ns5="http://www.w3.org/2005/Atom">
          <ns2:value>the driver class</ns2:value>
        </ns2:property>
      <!--The rest of the parameters required by a JDBC Connector-->
      </ns2:propertySheet>
    </ns2:touchpoint>
  </ns2:data>
  <content/>
</entry>

```

Note that the Touchpoint Server changes the ID of the entry to guarantee its uniqueness.

Furthermore, this time the Touchpoint Server response contains a Touchpoint Instance Destination Feed URL, which is needed for configuring the Touchpoint Destinations.

Next, we will add one Destination to the Initiator Touchpoint:

POST <Touchpoint Instance Destination Feed>

Body:

```

<entry xmlns="http://www.w3.org/2005/Atom">
  <scmp:destinations xmlns:scmp="http://www.ibm.com/xmlns/prod/scmp" >
    <scmp:destination>
      <scmp:request-out>Request-out URL</scmp:request-out>
    </scmp:destination>
  </scmp:destinations>
</entry>

```

The Touchpoint Server returns a response similar to:

201 Created

Location: <Touchpoint Instance Destination Entry URL>

Body:

```

<entry xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="http://a9.com/-/spec/opensearch/1.1/"
       xmlns:ns3="http://www.w3.org/1999/xhtml">
  <id>generated_id</id>
  <updated>2010-02-18T10:52:35.108+02:00</updated>
  <link href=<Touchpoint Instance Destination Entry URL> type="application/atom+xml;type=entry" rel="self"/>
  <link href=<Touchpoint Instance Destination Entry URL> rel="edit"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#resource" term="tp-destination"/>
  <ns2:destinations xmlns:ns2="http://www.ibm.com/xmlns/prod/scmp">
    <ns2:destination>

```

```

<ns2:request-out>Request-out URL</ns2:request-out>
</ns2:destination>
</ns2:data>
</entry>

```

At this point, the Initiator Touchpoint Instance starts executing.

Intermediary Touchpoint Instance

The steps needed to create a Touchpoint Instance in this Role are a combination of the ones for the Provider and Initiator roles.

First, we create a Touchpoint Instance Entry resource. This time the Touchpoint Instance Feed URL is concrete - `http://<tp_server_host>:<tp_server_port>/<context_root>/tp-node/default/tp-type/virtual_Intermediary/tp-inst`.

`POST <Touchpoint Instance Feed URL>`

Body:

```

<entry xmlns="http://www.w3.org/2005/Atom">
  <id>some ID</id>
  <title>Intermediary Touchpoint Instance</title>
  <author><name>author_name</name></author>
  <content/>
  <category term="intermediary-tp"
    scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-role" />
  <scmp: data xmlns:scmp="http://www.ibm.com/xmlns/prod/scmp" >
    <scmp: touchpoint>
      <scmp: propertySheet>
        <!--No parameters are required-->
      </scmp: propertySheet>
      <scmp: admin-state>enabled</scmp: admin-state>
    </scmp: touchpoint>
  </scmp: data>
</entry>

```

The Touchpoint Server returns a response similar to:

```

201 Created
Location: <Touchpoint Instance Entry URL>
Body:
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="http://a9.com/-/spec/opensearch/1.1/"
  xmlns:ns3="http://www.w3.org/1999/xhtml">
  <id>generated_ID</id>
  <updated>2010-02-18T11:20:00.546+02:00</updated>
  <title>Intermediary Touchpoint Instance</title>
  <link href=<Touchpoint Instance Entry URL>" type="application/atom+xml;type=entry" rel="self"/>
  <link href=<Touchpoint Instance Entry URL>" type="application/atom+xml;type=entry" rel="edit"/>
  <link href=<Touchpoint Type Entry URL>" type="application/atom+xml;type=entry"
    rel="http://www.ibm.com/xmlns/prod/scmp#resource-type"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-type" term="virtual://Intermediary"/>
  <link href=<Touchpoint Status Entry URL>" type="application/atom+xml;type=entry"
    rel="http://www.ibm.com/xmlns/prod/scmp#status"/>
  <link href=<Touchpoint Destinations Feed URL>" type="application/atom+xml;type=feed"
    rel="http://www.ibm.com/xmlns/prod/scmp#tp-destination"/>
  <author><name>author_name</name></author>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#touchpoint-role" term="intermediary-tp"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#resource" term="touchpoint"/>
  <ns2: data xmlns:ns2="http://www.ibm.com/xmlns/prod/scmp">
    <ns2: touchpoint>
      <ns2: admin-state>enabled</ns2: admin-state>
      <ns2: propertySheet/>
    </ns2: touchpoint>
  </ns2: data>
  <content/>
</entry>

```

Note that the Touchpoint Server changes the ID of the entry to guarantee its uniqueness.

Next, we will add one Destination to the Intermediary Touchpoint:

```
POST <Touchpoint Instance Destinations Feed>
```

Body:

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <scmp: data xmlns:scmp="http://www.ibm.com/xmlns/prod/scmp" >
    <scmp: destination>
      <scmp: request-out>Request-out URL</scmp: request-out>
    </scmp: destination>
  </scmp: data>
</entry>
```

The Touchpoint Server should return a response similar to:

201 Created

Location: <Touchpoint Instance Destination Entry URL>

Body:

```
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="http://a9.com/-/spec/opensearch/1.1/"
  xmlns:ns3="http://www.w3.org/1999/xhtml">
  <id>generated_id</id>
  <updated>2010-02-18T10:52:35.108+02:00</updated>
  <link href="<Touchpoint Instance Destination Entry URL>" type="application/atom+xml;type=entry" rel="self"/>
  <link href="<Touchpoint Instance Destination Entry URL>" rel="edit"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#resource" term="tp-destination"/>
  <ns2: data xmlns:ns2="http://www.ibm.com/xmlns/prod/scmp">
    <ns2: destination>
      <ns2: request-out>Request-out URL</ns2: request-out>
    </ns2: destination>
  </ns2: data>
</entry>
```

Finally, we will get the URL used for sending request to the Intermediary Touchpoint Instnace. As for the Provider Touchpoint, this is done through the Status Entry.

Send an HTTP GET request to the <Touchpoint Instance Status Entry URL>, available in the Touchpoint Instance Entry.

The Touchpoint Server returns a response similar to:

200 OK

Location: <Touchpoint Instance Status Entry URL>

Body:

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="http://a9.com/-/spec/opensearch/1.1/"
  xmlns:ns3="http://www.w3.org/1999/xhtml">
  <id>generated_id</id>
  <link href="<Touchpoint Instance Status Entry URL>" type="application/atom+xml;type=entry" rel="self"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#resource" term="touchpoint"/>
  <category scheme="http://www.ibm.com/xmlns/prod/scmp#aspect" term="status"/>
  <ns2: data xmlns:ns2="http://www.ibm.com/xmlns/prod/scmp">
    <ns2: touchpoint-status>
      <ns2: request-in>Touchpoint Intermediary Request-in URL</ns2: request-in>
      <ns2: op-state-available></ns2: op-state>
    </ns2: touchpoint-status>
  </ns2: data>
</entry>
```

Chapter 18. Tombstone Manager

Introduction

Tivoli Directory Integrator 7.1.1 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 the *Tombstone Manager* of Tivoli Directory Integrator that creates *tombstones* for each AssemblyLine and configuration as they terminate. Tombstones contain exit status and other information that you can request through the Server API. Tombstone Manager also:

- Displays the status of an entire Tivoli Directory Integrator configuration in an AMC status window.
- Ensures repeated runs of AssemblyLines within Action Manager, for example, every 24 hours.
- Provides status information to Server API clients about AssemblyLines that run asynchronously.

The Tombstone Manager API is documented in the Java API documentation; look for class `com.ibm.di.api.Tombstone`.

Configuring Tombstones

The creation of Tombstones for AssemblyLines and Config Instances is configured by means of check boxes in a number of screens in the Configuration Editor (CE), as well as a number of options in the `global.properties` or `solution.properties` files.

Once configured, your Configs contain the following switches:

At the configuration level:

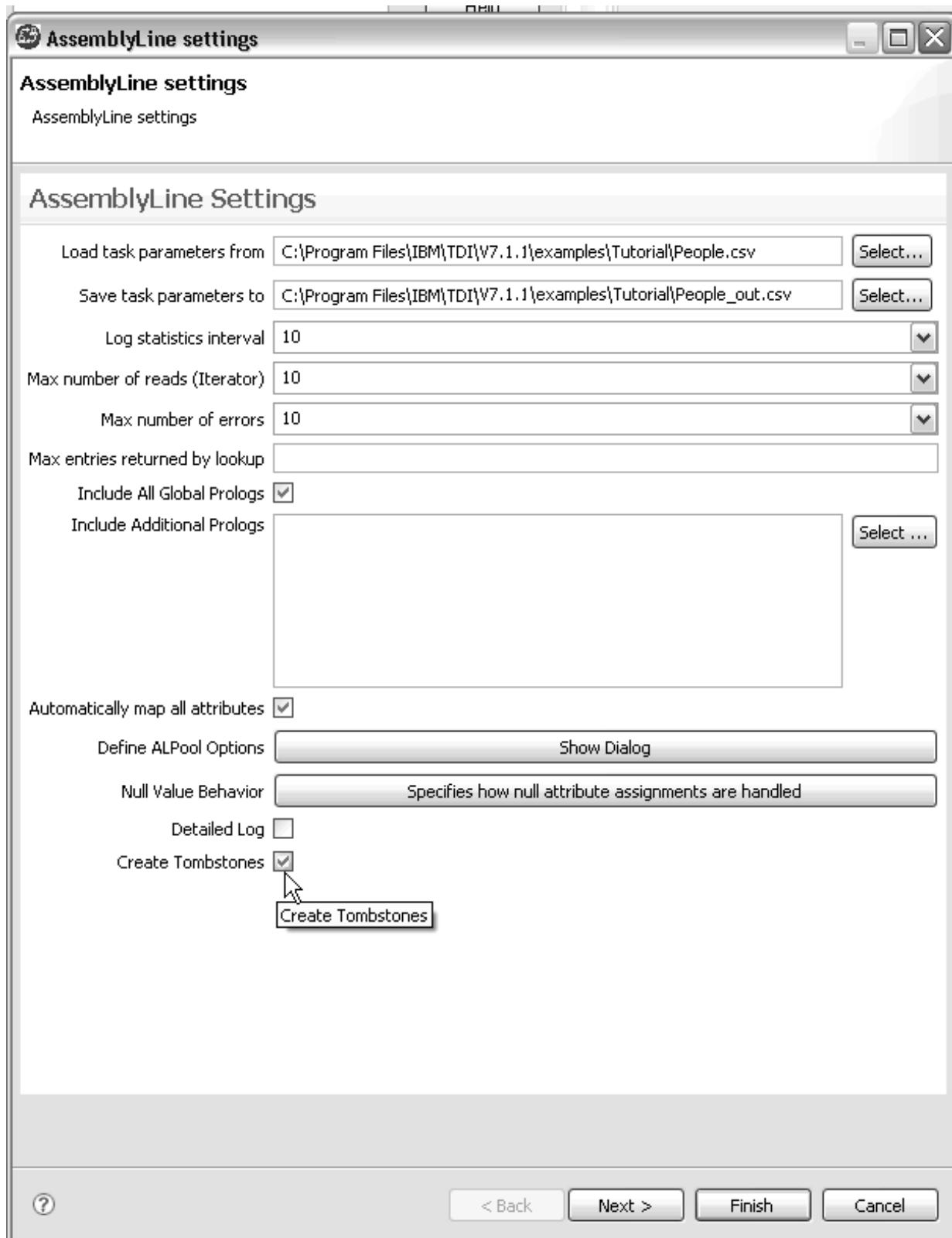
- Config switch: specifies whether tombstones are created or not for the Config Instance itself.
- All AssemblyLines switch: specifies whether tombstones are created for all AssemblyLines from this configuration.

At the AssemblyLine level:

A switch that specifies whether tombstones are created for that particular AssemblyLine. This switch is only taken into account when the "All AssemblyLines switch" at the configuration level is switched off.

Configuration Editor Configuration screen

You can configure tombstones creation for an AssemblyLine using the following AssemblyLine Configuration window. The Create Tombstone option is near the bottom of the window.



Checking **Create Tombstones** cause tombstones to be generated for this AssemblyLine when runs, even when the master switch for AssemblyLines is disabled.

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**.

A sample tombstone record could look like this:

Table 33. Tombstone record

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"
Statistics	[get:571] [add:571] [err:0]

The statistics returned can be one or more of the following attributes:

Table 34. Statistics returned in a Tombstone record

Attribute	Description
add	Total number of entries the AssemblyLine has added (performed by Connectors in AddOnly mode)
mod	Total number of entries the AssemblyLine has modified (performed by Connectors in Update mode)
del	Total number of entries the AssemblyLine has deleted. (performed by Connectors in Delete mode)
get	Total number of entries the AssemblyLine has retrieved (performed by Connectors in Iterator mode)
request	Total number of requests accepted when there is a Server mode Connector in the AssemblyLine
callReply	Total number of Call/Reply operations the AssemblyLine has executed (performed by Connectors in CallReply mode)
err	Total number of errors encountered
skip	Total number of entries the AssemblyLine has skipped entries
lookup	Total number of Lookup operations the AssemblyLine has executed (performed by Connectors in Update/Delete/Lookup mode)
ignore	Total number of entries the AssemblyLine has ignored (performed by Connectors in Update/Delta mode)
reconnect	Total number of times the AssemblyLine has attempted to reconnect to another client
exception	The exception text if the AssemblyLine terminated with an exception
getTries	Total number of times the AssemblyLine has attempted to retrieve an entry (performed by Connectors in Iterator mode)

Table 34. Statistics returned in a Tombstone record (continued)

Attribute	Description
getclientTries	Total number of times the AssemblyLine has attempted to get the next connected client (performed by Connectors in Server mode)
nochange	Total number of entries the AssemblyLine processed but left unchanged
branchtrue	Total number of Branch components executed by the AssemblyLine because their expression evaluated to true
branchfalse	Total number of Branch components skipped by the AssemblyLine because their expression evaluated to false
loopstart	Total number of Loop components executed by the AssemblyLine
loopcycles	Total number of cycles executed for all Loop components that had more than one cycle in an AssemblyLine
reconnectTime	Time in ms after last reconnect was attempted by the AssemblyLine

The Tombstone Manager

The Tombstone Manager 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”).

- The Tombstone Manager tracks Config Instances and AssemblyLines stop events.
- The Tombstone Manager uses the local Server API calls for registering for event notifications and receiving stop events for Config Instances and AssemblyLines.
- The Tombstone Manager uses the TDI System Store for data persistence.
- The Server API (documented in the Java API documentation) interfaces contain calls for querying the Tombstone Manager for various data, like AssemblyLine 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 executed through the Server API calls that provide all the information contained in the tombstone record.
- The tombstone record is deleted from the System Store either when an explicit cleanup Server API call is executed that deletes it, or when the logic for automatic deletion of old tombstone records collects it. Neither of these events is required, so theoretically, the tombstone record may live forever.

Tombstone Manager

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 live. When this property is present and contains an integer value greater than 0 the Tombstone Manager 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 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 create Tombstones for every AssemblyLine 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 are automatically deleted. Also when the total number of tombstone records reaches 50000, the oldest 25000 tombstone records is automatically deleted.

Chapter 19. Multiple TDI services

IBM Tivoli Directory Integrator as Windows Service

Introduction

In IBM Tivoli Directory Integrator 7.1.1 there is a mechanism that allows multiple Tivoli Directory Integrator 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.

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 is "ibmdisrv-test" and the Windows service display name is "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 Tivoli Directory Integrator Windows service configuration properties file see the "Configuring the service" on page 298 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 installation folder of the IBM Tivoli Directory Integrator is referred to as `root_directory`. See installer for Windows platforms.
2. Choose a solution folder that is 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 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 Tivoli Directory Integrator 7.1.1 version of the "ibmdiservice.exe" utility program any registered pre-Tivoli Directory Integrator 7.1.1 Windows service must be uninstalled and then the Tivoli Directory Integrator 7.1.1 windows service must be installed. This is necessary because the Tivoli Directory Integrator 7.1.1 windows service uses a different default name for the Windows service name – "ibmdisrv" as opposed to the pre-Tivoli Directory Integrator 7.1.1 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 you installed 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 (see uninstaller for Windows platforms).

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 three options to start the service:

- Restart the computer.
- Start the IBM Tivoli Directory Integrator service from the Windows Services window.
- Use the command line. See “Command line support” on page 304

Manual start and stop

You can manually start and stop the IBM Tivoli Directory Integrator service from the Windows Services window.

In the **Services** window 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**.

You can also use the command line; see “Command line support” on page 304.

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 window 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 installation of the log service.

Note: Before running the service, make sure this file is properly configured as described in this section.
The service could 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\TDI\V7.1.1`). 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.

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 V7.1.1 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.

servicename

Specifies a name that is used to form the Windows service name and the Windows service display name. The windows service name is set to the value of the `servicename` property prefixed with the "ibmdisrv-" prefix. The windows service display name is created by inserting the value of the `servicename` property between the brackets of the "IBM Tivoli Directory Integrator ()" expression.

For example, if the property value is "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 Windows service name is "ibmdisrv" and the default Windows service display name is "IBM Tivoli Directory Integrator".

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.

autostart

Specifies whether the Windows service starts automatically on Windows start-up or whether it has to be started manually. The valid values for this property are `true` and `false`. A value of `true` specifies that the Windows service is started on Windows start-up and a value of `false` specifies that the service has to be started manually. If this property is not present or has no value, then the default value of `true` is used.

This property is used during Windows service installation and changing it after the Windows service has been installed has no effect.

controlledshutdown

Specifies whether the Windows service will terminate the server gracefully or will hard kill the server process. The valid values for this property are "true" and "false". A value of "true" specifies

that the Windows service will stop the TDI server gracefully and a value of "false" indicates that the server process will be hard killed. If this property is not present or has no value, then the default value of "false" is used.

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\V7.1.1\jvm\jre\bin;  
C:\Program Files\IBM\TDI\V7.1.1\libs;  
ibmdirroot=C:\Program Files\IBM\TDI\V7.1.1  
configfile=rs.xml  
assemblylines=AssemblyLine1,AssemblyLine2  
cmdoptions=-d  
debug=false  
controlledshutdown=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 describe here. Some of the information presented could be used to construct scripts using 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 command.
- 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 computers N is 9, on RedHat Linux N is 6, and so on.). 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 Tivoli Directory Integrator 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 not be restarted. All subsequent reads of the /etc/inittab file while the init command is in the same run level cause the init command to ignore this entry.

- The **Command** field specifies the shell command to run.

Here are three example TDI-related entries in /etc/inittab:

```
tdi1::once:/opt/IBM/TDI711_1/ibmdisrv -c "/opt/IBM/TDI711_1/myconfigs/rs1.xml" -r "testAL1"
tdi2::once:/opt/IBM/TDI711_2/ibmdisrv -c "/opt/IBM/TDI711_2/myconfigs/rs2.xml" -r "testAL2"
tdi3::once:/opt/IBM/TDI711_3/ibmdisrv -c "/opt/IBM/TDI711_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 for system startup. That is why the information provided here covers the main issues of starting IBM Tivoli Directory Integrator 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://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.files/doc/aixfiles/inittab.htm>

Graceful shutdown:

On UNIX systems we always perform a graceful shutdown of the service. This is achieved by a shutdown hook added to the Java Runtime. That way, when the server is stopped with either SIGINT or SIGTERM signals, this hook is executed and the server is gracefully terminated.

Another advantage of this approach is that this hook will be invoked on all platforms when the server is stopped by pressing CTRL+C in its console window.

Note: You can also specify an external program to be started from within the JVM shutdown Hook. This external program is configured using an optional property in the **global.properties** or **solution.properties** file: **jvm.shutdown.hook**. If configured the external program will be started right after the server has shutdown gracefully.

IBM Tivoli Directory Integrator as z/OS Service

USS process

In order to start an instance of the TDI server when the Unix System Service of z/OS is started, you must have the HFS data set for Tivoli Directory Integrator mounted before the Unix Service is initialized.

This could be done by adding a record for mounting the HFS data set to the BPXPRM01 data member of the USER.PARMLIB data set.

For example the record for mounting the HFS data set might look like this:

```
MOUNT FILESYSTEM('TDI.V7R1M0.HFS') TYPE(HFS) MODE(READ) MOUNTPOINT('/usr/lpp/itdi')
```

After mounting the HFS data set permanently, you should add a record in the /etc/rc file that starts an instance of the Tivoli Directory Integrator Server. The record in the /etc/rc file must specify the solution directory.

For example:

```
/usr/lpp/itdi/ibmdisrv -s /u/musala/tdi_solutions -c rs.xml -r a1
```

In order for the changes in the USER.PARMLIB(BPXPRM01) data member and the /etc/rc file to take effect, the z/OS computer needs to be re-IPLED (rebooted).

Normal z/OS started task

Tivoli Directory Integrator ships with an example in order to illustrate how a Tivoli Directory Integrator instance can be started as a normal z/OS task. It consists two parts: shell script and JCL.

Shell script: iditask.sh:

The shell script iditask.sh can be found in the *TDI_install_dir/bin* folder (by default *usr/lpp/itdi/bin*). It contains the actual shell command for starting a TDI instance as well as exports several environment variables, which are needed by the further processing of the script by the z/OS native program.

The script sets the `JAVA_HOME` environment variable to the path of the Java 6 JVM, which is the required version to run the TDI server, and exports the `LIBPATH` variable by adding the TDI installation directory to it.

In order to run any child processes in the same address space as the started task the environment variable `_BPX_SHAREAS` is set to YES in the script.

The script implements logic to distinguish the actual installation directory of TDI under which it is visible in USS. In Tivoli Directory Integrator 7.1.1, the `iditask.sh` script invokes the `ibmdisrv` command and passes it all arguments received from the ADITASK JCL script when invoked. The actual command doing this is as follows:

```
$TDI_DIR/ibmdisrv $*
```

where `TDI_DIR` specifies the TDI installation directory visible in USS.

This approach reduces the customization needed when starting the TDI server as a z/OS task. Now if you want to change the server start command or logging options, only the ADITASK JCL script needs to be modified. This way the `iditask.sh` doesn't need to be modified, it also avoids any problems when the TDI directory is read-only.

JCL: ADITASK:

The second part of the mechanism is the ADITASK JCL, which invokes the shell script described above and starts Tivoli Directory Integrator as a normal z/OS task in its own address space.

It resides in MVS in the adi.HADI700.F1 library, where adi stays for the high level qualifier of the product chosen by the user (usually ADI or TDI), as well as in the corresponding AADIJCL and SADIJCL libraries (for example, TDI.V7R1M0.AADIJCL). From a legacy perspective it is better to first copy it into an user's CNTL and submit from there.

The ADITASK uses the BPXBATCH native utility to start the shell script.

Note that it specifies the predefined location of "iditask.sh" - /usr/lpp/itdi/bin/iditask.sh. In case TDI is installed in a location different from the default (/usr/lpp/itdi) or the script is moved to another location after the installation ADITASK JCL must be edited in order to function.

The ADITASK redirects STDOUT, STDERR and LOGOUT to the standard output of the task (SYSOUT). Thus every message sent to the console is logged under the task's name in SDSF. "Configuring the TDI task to log to its SYSOUT" on page 309 The behavior can be changed, so that the relevant messages are stored in an intermediate USS file or another MVS data set.

Below is an example as to how STDOUT can be redirected from SYSOUT to the USS file /usr/lpp/itdi/stdout.txt. In the ADITASK just replace the statement:

```
//STDOUT DD SYSOUT=*
```

with the following ones:

```
//STDOUT DD PATH='/usr/lpp/itdi/stdout.txt',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
```

TDI autostarting:

There is an additional step in order to automatically start the task with the start of the system. For this purpose the COMMNDxx member in the SYS1.PARMLIB data set, which is a mechanism of having console start commands invoked during the bring-up of a system image, must be edited. By having a "started task" named for example ADITASK defined as an ADITASK member in the SYS1.PROCLIB data set, the user can get that task started during system startup by having a CMD='S ADITASK' in the COMMNDxx member.

An alternative is to add the task to USER.PROCLIB and editing the COMMNDxx member of USER.PARMLIB or to use another library concatenated to the Job Entry Subsystem's (JES2 or JES3) PROCLIB allocation in case the user does not have write access to SYS1 members.

By this means the ADITASK can be started from the SDSF menu with the standard z/OS START command. For example:

```
/S ADITASK
```

IBM Tivoli Directory Integrator as i5/OS Service

Tivoli Directory Integrator 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 is called by the STR/ENDTCPSVR command.

The following are the steps to manually start a Tivoli Directory Integrator service which start an AssemblyLine "AL" in Config file rs.xml .

1. Create Library for Tivoli Directory Integrator:

```

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(STROSH 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 Tivoli Directory Integrator TCP/IP server using the **ADDTCPNSVR** command with name **TDISERVER**:

```

ADDTCPNSVR SVRSPCVAL(*TDISERVER)
PGM(TDILIB/TDIPGM)
SVRNAME(TDISERVER)
SVRTYPE(TDISERVER) AUTOSTART(*NO)

```

6. Start the TDI TCP/IP server:

```
STRTCPNSVR SERVER(*TDISERVER)
```

For more information on the i5/OS (OS400) (V5R4) and commands that have been demonstrated above, see: <http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=/clfinder/finder.htm>

Note: The graceful shutdown option is not supported on i5/OS systems.

Command line support

Tivoli Directory Integrator 7.1.1 provides a script for starting and stopping the Tivoli Directory Integrator service on Windows and UNIX systems. The script is located in the *TDI_install_dir/bin* directory, and is named *servicemgr.bat(sh)*.

The usage of the script is as follows:

```
servicemgr service_name start|stop
```

where:

service_name

is the name of the service. For Windows systems this is *ibmdisrv* by default or the value of *servicename* property in the *ibmdiservice.props* file. For UNIX systems this is the identifier field from the */etc/inittab* file.

start|stop

the desired command to perform on the service.

Chapter 20. z/OS environment Support

The full set of IBM Tivoli Directory Integrator components is available in the native z/OS environment (version 1.7 and 1.8), with IBM JVM 1.6, except for those components that require native libraries, like the Windows Users and Groups Connector and the Domino Change Detection Connector. Conversely, the z/OS Command Line Connector is available on z/OS only.

You can start an instance of the Server by executing the startup shell script `usr/lpp/itdi/ibmdisrv` residing under the Unix System Services. Also see “IBM Tivoli Directory Integrator as z/OS Service” on page 302 for information about automatic startup of a Tivoli Directory Integrator Server, either as a USS process or a normal z/OS started task.

The user whose identity the Tivoli Directory Integrator Server runs under needs an OMVS segment definition in his profile specifying that at least 200MB operating memory is available.

The z/OS TSO Command Line Function Component is of particular relevance for the z/OS environment. It is able to run 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 Configuration Editor and AMC, however, are not supported natively on z/OS; instead, you should use remote management options, like

- The Remote Configuration Editor. Run the Configuration Editor on a supported platform, and access Config files on z/OS using a configured Config Instance on z/OS.
- The Administration and Monitoring Console
- Applications that use the remote Tivoli Directory Integrator Server API.

For installation instructions and hardware requirements on z/OS, see the Program Directories physically shipped with the product:

- Program Directory for IBM Tivoli Directory Integrator General Purpose Edition (GI11-9329-00: TDI_GPE.PDF)
- Program Directory for IBM Tivoli Directory Integrator Identity Edition (GI11-9328-00: TDI_IE.PDF)

Post install configuration

Using MQe for system queue

From IBM Tivoli Directory Integrator v7.1.1 onwards, ActiveMQ is the default system queue. To use MQe as a system queue, you invoke:

```
TDI_install_dir\jars\plugins\mqeconfig.bat(.sh) TDI_install_dir\jars\plugins\mqeconfig.props create server
```

If you do not want to use the system queue, set the `systemqueue.on` property of the `global.properties` or `solution.properties` file to false.

Default encoding different than IBM-1047

The text files as installed by SMP/E are encoded using the IBM-1047 encoding. If the default character encoding on your system is different from IBM-1047 you need to perform the following post-installation steps before the IBM Tivoli Directory Integrator installation directory is made read-only in order to convert the text files to your default encoding:

1. Change the current working directory to `"/usr/lpp/itdi/tools"`.
2. Issue the following command:

- ```
iconv -f IBM-1047 -t YOUR_DEFAULT_ENCODINGencconvz > ../encconvz
```
3. The previous command should have created a shell script called "encconvz" in the "/usr/lpp/itdi" folder
  4. Run the following command to change the mode of the file:  
`chmod u+x encconvz`
  5. Run the encconvz script.

## JDK 5.0 not located at /usr/lpp/java/J5.0

Ensure that either the JRE\_PATH or the JAVA\_HOME environment variable is set accordingly. For example: If your Java SDK is installed in "/usr/lpp/java/MyJava5.0", then

```
JAVA_HOME=/usr/lpp/java/MyJava5.0
```

or

```
JRE_PATH=/usr/lpp/java/MyJava5.0/bin
```

## Running Tivoli Directory Integrator

Since the install directory of Tivoli Directory Integrator (/usr/lpp/itdi) is read-only, Tivoli Directory Integrator must be run from a solution directory (as explained in the documentation) which is different from the install directory. That is why before running Tivoli Directory Integrator, you should create a solution directory (which is just a normal directory different from the install directory), make this solution directory the current working directory and then launch the Tivoli Directory Integrator server. You can follow these steps in order to accomplish the creation of a solution directory and launching the Tivoli Directory Integrator server:

1. `mkdir solution_dir`
2. `cd solution_dir`
3. `/usr/lpp/itdi/ibmdisrv TDI_PARAMETERS`

where *TDI\_PARAMETERS* are just the normal parameters the Tivoli Directory Integrator server reads from the command line. The first time the server runs this way, it will populate the solution directory with a number of configuration files based on those in the installation directory; you can now customize the files in the solution directory for your own needs.

## Reading License Files

All license files are UTF-8-encoded so that NLS characters are preserved. Most OS tools can only read files encoded using the default (native) character encoding. That is why before you can read a license file using a general purpose tool, you need to convert its encoding. You can do that using the `iconv` utility in the following way:

```
iconv -f UTF-8 -t YourDefaultCharacterEncoding LicenseFile DestinationFile
```

where *YourDefaultCharacterEncoding* is the native character encoding of your z/OS system, *LicenseFile* is the license file you wish to convert and *DestinationFile* is the text file that will contain the contents of *LicenseFile*, but encoded using *YourDefaultCharacterEncoding*.

After this command completes you should be able to open and read *DestinationFile* (provided you have specified the correct encoding and that encoding supports all characters in *LicenseFile*).

Another option for reading the license files is to transfer them in BINARY mode via FTP to a system which can read UTF-8-encoded text files and supports the NLS characters contained in the license file you wish to read.

---

## Using the Remote Configuration 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 Configuration Editor. See "Using the Remote Configuration Editor" on page 131 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 must be uploaded (using FTP) in binary mode to the z/OS computer.
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 the other way round.

Any file created on a Windows or Unix computer (for example, a properties file, a text file, and so on.) that needs to be read by TDI running on a z/OS system 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. The tdisrvctl CLI remote utility is installed in the *TDI\_root\_directory/bin* directory. See "Command Line Interface – tdisrvctl utility" on page 189.

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.appenders.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 computer'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>** is 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 is 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 (that is, global.properties, Log4J.properties, and so on) 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.

---

## Using ASCII mode

The ibmdisrv startup script starts the Tivoli Directory Integrator server without altering its default encoding, which on z/OS is EBCDIC (IBM1047). In order to run the server on z/OS in ASCII mode you must start it using the ibmdisrv\_ascii startup script. This script starts the server with its default encoding set to ASCII (ISO-8859-1).

**Note:** In ASCII mode, the 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.

### Encoding of `solution.properties`:

Altering the default encoding of the Tivoli Directory Integrator 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 Tivoli Directory Integrator 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 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 must have all the required properties for your solution in the `solution.properties` file in your Solution Directory.

### Log files:

When the 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 have to first convert their encoding to the native encoding on z/OS, which is EBCDIC (IBM1047).

### Console output:

When the 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).

## Configuring the TDI task to log to its SYSOUT

The Tivoli Directory Integrator product implementation on z/OS provides the ability to redirect the information from the intermediate USS `ibmdi.log` file within the related HFS file to the TASK SYSOUT of the TDI started task.

Configuring the TDI task to log to its SYSOUT allows z/OS users to have all the relevant information in a single place; it also allows people not fully skilled on Unix System Services to find the logs using SDSF. In addition the full *sysout* (including product messages) is saved and kept using the normal tool. Using this approach ensures that each start of the TDI TASK stores the information in its own *sysout* thus preventing any log replacement.

The redirection of the TDI server logs consists of two configuration steps: editing the `log4j.properties` file and the JCL script starting TDI as z/OS task (ADITASK; also see “IBM Tivoli Directory Integrator as z/OS Service” on page 302). This approach relies on the ability of newer releases of z/OS including the oldest supported by TDI - version 1.6 - to route STDOUT and STDERR to SYSOUT instead of an ‘intermediate’ USS file. For this purpose all the needed log information should be first made visible from within the standard console. In order to take advantage of this feature the two steps described below should be performed.

### Modifying `log4j.properties`

This step is required in order to redirect the log messages to the standard console output. For this purpose the `log4j.properties` file used by the TDI instance must be edited to add a console appender and to use it as default logger (to ensure that all log messages are stored in the task SYSOUT) or specify it as a logger for certain TDI objects like ALs, configurations and the like, so that only relevant information is added.

For example, to add and configure a console appender as default logger:

```
Here is an example on how to make a logger that logs to the console
log4j.appenders.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appenders.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appenders.CONSOLE.layout.ConversionPattern=%d [%t] %-5p - %m%n
log4j.appenders.CONSOLE.encoding=IBM-1047
```

Changing the root logger to the CONSOLE appender:

```
log4j.rootCategory=INFO, CONSOLE
```

With this modification the log information is now routed to the standard console and can be further manipulated in the MVS environment.

### Modifying the starting JCL

The second step involves modification of the JCL responsible for starting TDI as normal z/OS started task, so that the messages received from the console are routed to the SDSF visible logs of the task. Note that if the ADITASK JCL shipped as an example with TDI is used, this step is redundant, since its default behavior is to log there.

Edit the JCL, which starts the TDI server as standard z/OS started task, to redirect the log output to the task SYSOUT. This could be done by simply adding the lines:

```
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
```

or

```
//STDOUT DD SYSOUT=(,)
//STDERR DD SYSOUT=(,)
```

Since the z/OS system takes advantage of the Log4J options, it stores all the messages with status ERROR and FATAL in the STDERR associated with the TASK and the others - DEBUG, INFO,

and so on. in the STDOUT. The length of the created records in this manner is not fixed to 133 characters, therefore extremely long or multiline messages can be saved without truncation.

The information can be easily routed or copied to another data set or intermediate USS file. By this means it can be stored both in MVS and USS environment.

Below is an example showing how the STDOUT can be redirected back to the `ibmdi.log` file from the JCL:

```
//STDOUT DD PATH='/usr/lpp/itdi/logs/ibmdi.log',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
```

**Note:** By formatting the messages a non-readable character appears between some of the components of the message, for example, the Date and the message ID, which is due to a conversion issue when transferring characters from ASCII to EBCDIC. When using the characters "[" and "]" in the pattern, they are not translated properly in the z/OS native encoding. The "[" character is replaced by the "Y" and "]" is converted to ".

---

## 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 Tivoli Directory Integrator 7.1.1 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 Tivoli Directory Integrator 7.1.1 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.

**AES** Shorthand for Advanced Encryption Standard. AES is an encryption algorithm for transmitting sensitive (but unclassified) content by U.S. Government agencies.

#### Adapter

*Adapter* is a word 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 used by the AssemblyLine Connector, which offers mode settings reflecting these operations<sup>2</sup>.

**AL** Shorthand for AssemblyLine.

#### Administration and Monitoring Console (AMC)

AMC is a browser-based console for managing and monitoring solutions. AMC Version 3, which is part of the Tivoli Directory Integrator 7.1.1 release, runs inside the Integrated Systems Console (ISC). Each AMC version is designed to work with a specific release of TDI and may be incompatible with other versions. AMC v.3 is designed for Tivoli Directory Integrator 7.1.1, however, it also works with TDI 6.1.X and TDI 6.0 (albeit with some restrictions). 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

---

2. AL Operations are also accessible via the AssemblyLine FC.

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.

### **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 AL's 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<sup>3</sup>. 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.

### **AssemblyLine Sequence**

The AssemblyLine Sequence is used to run AssemblyLines together with a minimum of conditional capabilities.

---

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**

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.

## **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 is 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 does this make your solution easier to read and maintain (and you can step through with the AL Debugger!), but your solution benefits 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. Tivoli Directory Integrator 7.1.1 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 V7.1.1 Reference Guide*.

**CEI** The IBM Common Event Infrastructure. See "The Common Event Infrastructure", in *IBM Tivoli Directory Integrator V7.1.1 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.<sup>4</sup>.

**CLI** Command-line Interface, such as the `tdisrvctl` utility.

**cipher** A cipher is any method of encrypting text (hiding its readability and its meaning). The resulting encrypted text message is called ciphertext.

**ciphertext**

Ciphertext is encrypted text, the result of applying a cipher, or an encryption.

## 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 compares 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 Configuration Editor.

## Config Browser

This is the tree-view window at the top left-hand part of the Configuration 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.

## Configuration 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 run.

## 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.

## conn Entry

This is the local Entry object maintained by a Connector or Function. The conn Entry is used as a 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

---

4. For LDAP there is also a special kind of modify operation where the directory entry has been moved in the tree: *modrdn*, that is, a "renamed" entry.

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*.<sup>5</sup> 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 V7.1.1 Reference Guide* for more information. See also "Connectors" in *IBM Tivoli Directory Integrator V7.1.1 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 V7.1.1 Users Guide* for more information.

### Delta mode (for Connectors)

This Connector mode is used to 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 does not work with Entries without a valid delta operation code. See "Deltas and compute changes" in *IBM Tivoli Directory Integrator V7.1.1 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<sup>6</sup>. 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 V7.1.1 Users Guide*.

**Derby** Apache Derby (previously known as Cloudscape) is a small footprint relational database implemented entirely in Java. Derby v10.5.3 is shipped as the default System Store for TDI.

**DES** Short for Data Encryption Standard. DES is a widely-used method of data encryption using a secret key. DES is superseded by the Advanced Encryption Standard (AES).

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.

## 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 ( , ).

- ECB** Short for Electronic Code Book. Electronic Code Book (ECB) is a method of operation for a block cipher. In ECB, each possible block of plaintext has a defined corresponding ciphertext value and the other way around. The same plaintext value always results in the same ciphertext value. Electronic Code Book is used when a volume of plaintext is separated into several blocks of data, each of which is then encrypted independently of other blocks. Moreover, Electronic Code Book can create a separate encryption key for each block type.

## Easy ETL

'ETL' stands for Extract, Transform and Load, and boils down to getting data from one place, changing it as needed and then putting it someplace else. 'EasyETL' is a feature of Tivoli Directory Integrator that lets you do this quickly and interactively in just a few keystrokes.

- 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"<sup>7</sup>. 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 uses 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<sup>8</sup>. This is the outer line of defense<sup>9</sup>. 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:

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.

```

try {
 myObj = someFunctionCallThatCanThrowAnException();
} catch (ExceptionThrown) {
 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.

### 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.

- FIPS** Short for Federal Information Processing Standard. TDI uses FIPS 140-2, a standard that defines requirements for cryptographic modules that handle sensitive information.

- 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 run 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 Configuration 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

HyperText Markup Language. 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** HyperText Transfer Protocol. The protocol in use for the WorldWide Web, another protocol on top of TCP.

### 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 causes any Iterators in the Flow section to skip on this cycle.

### Integration Framework (IF)

The Integration Framework (IF) is a set of applications that facilitates integration between the system and framework applications. IF is a part of base Tivoli Process Automation Engine and is

available in all major products that use Tpae. For example, Maximo Asset Management (MAM), Tivoli Service Request Manager (TSRM), and so on.

The IF is an integral part of Tpae. It is an XML based integration framework and supports both XML and delimited files. IF allows synchronization and integration of data between an external system and applications that use the Tpae common architecture and run under an application server. Using IF, you can exchange data synchronously and asynchronously, using a variety of communication protocols.

#### **Iterator**

A Connector mode<sup>10</sup> 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.

#### **Java Virtual Machine or 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 computer within the computer.

#### **Java API documentation (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 Java API documentation can be viewed by selecting the **Help -> Welcome** screen, **JavaDocs** link from the Configuration Editor.

You can download and unzip Javadocs for any package into the docs/api directory of your installation to get the complete documentation for java classes that are not documented in TDI Javadocs.

#### **JavaScript**

The language you can use to fine tune the behavior of your AssemblyLines. Tivoli Directory Integrator 7.1.1 uses the IBM JSEngine.

**JMS** Java Messaging Service. A standard protocol used to perform guaranteed delivery of messages between two systems.

**JNDI** Java Naming and Directory Interface. See "JNDI Connector", in the *IBM Tivoli Directory Integrator V7.1.1 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** Lightweight Directory Access Protocol. 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.

---

10. Connectors running in Iterator mode are often referred to as "Iterators".

## **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**.

**Mode** Connectors have a mode setting that determines how this component participates 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 varies. See "Connectors" in *IBM Tivoli Directory Integrator V7.1.1 Reference Guide* for more information about Connector modes.

## **Null Value Behavior**

This term refers to how TDI deals 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 **More** button in the button bar at the top of the map and selecting **Null behavior**; or for a specific Attribute via the **Null behavior** context menu item in the Assignment column 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 Tivoli Directory Integrator.

**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.

## **plaintext**

Plaintext is unencrypted text. In cryptography, plaintext is ordinary readable text before being encrypted into ciphertext or after being deciphered.

**Prolog** A set of Hooks that, if enabled, are run during the Assemblyline Initialization. You can also define Global Prologs: Script Components 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 the 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.<sup>11</sup>.

## 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, Tivoli Directory Integrator Server settings that are kept in a file called `global.properties` in 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<sup>®</sup>)

In LDAP terms the name of an object that is unique relative to its siblings. RDNs have the form *attribute name=attribute value*. For example,

`cn=John Doe`

## Resource Library

A simple method for sharing AssemblyLines and components between Configs. In the Configuration Editor, the "Resources" navigator appears just below the Config Browser.

**RMI** Remote Method Invocation; a way of making procedure or method calls on a remote system using a network communication channel. In Tivoli Directory Integrator, used by the Remote API functionality.

**RSA** RSA is an internet encryption and authentication system that uses an algorithm developed by Ron Rivest, Adi Shamir, and Leonard Adleman. The encryption system is owned by RSA Security. RSA is an algorithm for public-key cryptography, suitable both for signing and for encryption.

## 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 V7.1.1 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. In addition to appearing in the Scripts library folder of the Config Browser<sup>12</sup>, Scripts can be dropped anywhere in the Flow section of an AssemblyLine.

---

11. Note that an Entry object can also hold *properties* (in addition to Attribute and delta operation codes) and these can be accessed using the `getProperty()` and `setProperty()` methods of the Entry class.

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.

## **Script Engine**

The component that interprets the Java scripts written inside a TDI Config. The IBM jsEngine is used by Tivoli Directory Integrator 7.1.1, which replaces Rhino from the previous releases.

## **Scheduler**

Use the TDI Scheduler to automatically start an AssemblyLine or Sequence specified in the configuration information, at a predefined times.

## **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.

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 run 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, Derby databases, properties files, keystores and so forth. The solution directory is selected when you install Tivoli Directory Integrator, and the filepaths used in your solution can be relative to this folder. The solution directory can be explicitly specified when you start the Configuration 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.

## **Solution View**

This term is used in the context of AMC to describe how a particular Config appears in the management screens of AMC. A Solution 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.

**SSL** Secure Socket Layer; a protocol used in Internet communications to encrypt data such that if someone were 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, Tivoli Directory Integrator uses it (if configured that way) to talk between different Tivoli Directory Integrator 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 participates 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 Apache ActiveMQ, but can be configured to use the bundled MQe (WebSphere MQ Everyplace) or any other JMS-compliant messaging systems. TDI provides a SystemQueue Connector to help you use this feature.

For more information about the System Queue and how to enable it, see the "System Queue" chapter in the *IBM Tivoli Directory Integrator V7.1.1 Installation and 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 Derby product (previously known as **CloudScape**) is used. See <http://db.apache.org/derby> 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** Transmission Control Protocol, 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 Defense, 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 Tivoli Directory Integrator.

### Tivoli Directory Integrator Dashboard

Tivoli Directory Integrator Dashboard can be used to install, configure, deploy, and monitor data integration solutions. You can also use Dashboard to create and configure EasyETL solutions (ETL stands for Extract, Transform, and Load).

### 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".

### Tivoli Process Automation Engine (Tpae)

The Tivoli Process Automation Engine (Tpae), also known as Base Services, is a collection of core Java classes and is used as a base to build Java applications. The Integration Framework, a Tpae feature, contains standard integration objects (Object Structures and interfaces) and outbound/inbound objects.

## Tombstone

A record or trace showing that an AssemblyLine, an EventHandler or Config has terminated. Configured through the Tombstone Manager in the CE. The trace includes a timestamp and the AL exit status. The 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<sup>13</sup>, and if found it modifies this entry. If no match is found then a new entry is added instead. See also Computed Changes.

**UDP** User Datagram Protocol. 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** Unified Resource Locator. 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 V7.1.1 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 Tivoli Directory Integrator 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<sup>14</sup>. 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 does 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 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 Xtensible Markup Language. A general purpose markup language (See also HTML) for

---

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.

14. Note that the "Work Entry" window shown in the Configuration Editor is actually a list of all Attributes that appear in Input Maps or in the Loop Attribute field of Loops in the AssemblyLine.

*creating* special-purpose markup languages, and also capable of describing many types of data IBM Tivoli Directory Integrator uses XML to store Config files.

---

## Appendix B. 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 or 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”
- “jlog.properties” on page 327
- “derby.properties” on page 328
- “global.properties” on page 328

Properties set in any of those files form a baseline for the entire IBM Tivoli Directory Integrator installation for all users on that computer. 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 overrides anything set in any of the global installation property files mentioned above. Furthermore, 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 Configuration Editor or the Server
- By specifying the -s parameter to the ibmditk script to start the Server. This takes precedence over setting *TDI\_SOLDIR*.

If *TDI\_SOLDIR* equals the installation directory, the behavior is like in older versions of Tivoli Directory Integrator: all property files are 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 Tivoli Directory Integrator Server, it makes a copy of all the property files into your Solutions Directory (it does 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 Tivoli Directory Integrator.

**Note:** The file *global.properties* is 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 are copied under their own name.

In addition, if your Solution Directory was setup during product installation using the Tivoli Directory Integrator installer, the setup will contain a working System Queue setup. If the Solution Directory is created by any other means (manually, or by the Server by using the -s option) then you will either have to disable the System Queue in your *solution.properties* file, or setup a System Queue yourself – see “System Queue Configuration” on page 143.

---

### Log4J.properties

This file sets a baseline for the log-strategy for the server (ibmdisrv).

Log options configured in the Logging tab in the Configuration 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=logs/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

#Example settings for changing the default logger

ROLLING FILE SIZE APPENDER
##RollingFileAppender rolls over log files when they reach a certain size specified by the
##MaxFileSize parameter

#log4j.appender.Default=org.apache.log4j.RollingFileAppender
#log4j.appender.Default.File=logs/ibmdi.log
#log4j.appender.Default.Append=true
#log4j.appender.Default.MaxFileSize=10MB
#log4j.appender.Default.MaxBackupIndex=10
#log4j.appender.Default.layout=org.apache.log4j.PatternLayout
#log4j.appender.Default.layout.ConversionPattern=%d{ISO8601} %-5p [%c] - %m%n

DAILY OUTPUT LOG4J SETTINGS
With the DailyRollingFileAppender the underlying file is rolled over at a user chosen frequency.
##The rolling schedule is specified by the DatePattern option

#log4j.appender.Default=org.apache.log4j.DailyRollingFileAppender
#log4j.appender.Default.file=logs/ibmdi.log
#log4j.appender.Default.DatePattern='yyyy-MM-dd
#log4j.appender.Default.layout=org.apache.log4j.PatternLayout
#log4j.appender.Default.layout.ConversionPattern=%d{ISO8601} %-5p [%c] - %m%n

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.AssemblyLines/myAL=INFO, CONSOLE

Or you could log to myFile.log
log4j.logger.AssemblyLine.AssemblyLines/myAL=INFO, fileLOG

```

## jlog.properties

This file configures the JLOG-based Chapter 15, “Tracing and FFDC,” on page 209 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 ITDI 7.1.1
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 under ITDI_HOME directory.
The default value is set to true.
#
#-----
jlog.noLogCmd=true

#-----
Set listen port for JLOG's command server
#
If you want LogManager to listen on different port than the default one (9992) you should
uncomment the property jlog.logCmdPort and set it to the desired port. If not uncommented
the LogManager will listen on the default port - 9992.
#
#-----
#jlog.logCmdPort=9992

#-----
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 number 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 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 hierarchy 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

#-----
Configure the PDLogger for the Config Editor and attach the Listeners to it.
By default, no listeners are attached
#-----
jlog.logger.config-editor.level=FATAL
jlog.logger.config-editor.listenerNames=

```

---

## derby.properties

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

```

This is a sample properties file provided to show the proper format.
We're also setting one property which make sure that
Derby 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` (the CE) and `ibmdisrv` (the server) on startup. This file is read and applied before a file called `solution.properties` from your Solution Directory is read and applied.

### Note:

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 l url> to include other files

com.ibm.di.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

##
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

##
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/$soldir$/
TDISysStore;create=true
com.ibm.di.store.jdbc.driver=org.apache.derby.jdbc.ClientDriver
com.ibm.di.store.jdbc.urlprefix=jdbc:derby://localhost:1527/
com.ibm.di.store.jdbc.user=APP
#{protect}-com.ibm.di.store.jdbc.password=APP

#
Derby (Cloudscape) properties required for enabling authentication
#
derby.drda.startNetworkServer=true
derby.connection.requireAuthentication=true
derby.authentication.provider=BUILTIN
derby.database.defaultConnectionMode=fullAccess

#
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
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.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 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.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, 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
PSEN31 BUFFERPOOL BP32K LOCKSIZE LOB;CREATE AUX TABLE TBPSEN3 IN PSEN31
STORES {0} COLUMN ENTRY;CREATE INDEX PSIXE3 ON TBPSEN3
#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}

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_TIME TIMESTAMP, CREATED_ON TIMESTAMP, COMPONENT_NAME VARCHAR(1024),
CONFIGURATION VARCHAR(1024), EXIT_CODE INT, ERROR_DESCR VARCHAR(1024),
STATS LONG VARCHAR FOR BIT DATA, GUID VARCHAR(1024) NOT NULL, USER_MESSAGE
VARCHAR(1024), UNIQUE (ID, GUID))

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

server authentication
javax.net.ssl.trustStore=serverapi/testadmin.jks
{protect}-javax.net.ssl.trustStorePassword=administrator
javax.net.ssl.trustStoreType=jks

client authentication
javax.net.ssl.keyStore=serverapi/testadmin.jks
{protect}-javax.net.ssl.keyStorePassword=administrator
javax.net.ssl.keyStoreType=jks

##PKCS11 options
##Set the value of following properties to use PKCS11 enabled devices to
store TDI servers private key / certificate.
com.ibm.di.pkcs11cfg/etc/pkcs11.cfg
com.ibm.di.server.pkcs11=false
com.ibm.di.server.pkcs11.library=
com.ibm.di.server.pkcs11.slot=
{protect}-com.ibm.di.server.pkcs11.password=PASSWORD

Turns on java debug
javax.net.debug=true

java interpreter override

```

```

com.ibm.di.javacmd=
com.ibm.di.installdir=

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

Following properties modified in TDI 7.1 .Added property for
keystore password and keypassword
com.ibm.di.server.keystore
com.ibm.di.server.key.alias

api.keystore=testserver.jks
api.keystore.type=jks
api.key.alias=server
{protect}-api.keystore.password=server
{protect}-api.key.password=

Encryption properties added in TDI 7.1
com.ibm.di.server.encryption.keystore = testserver.jks
com.ibm.di.server.encryption.key.alias = server
com.ibm.di.server.encryption.keystoretype = jks
com.ibm.di.server.encryption.transformation = RSA

Web container
web.server.port=1098
web.server.ssl.on=false
web.server.ssl.client.auth.on=false
web.server.session.timeout=300

Touchpoint Server properties
tp.server.on=false
tp.server.config=etc/tp.xml
tp.server.auth=false
tp.server.auth.realm=Tivoli Directory Integrator Touchpoint Server

Dashboard properties
##
dashboard.on=true
dashboard.templates.folder=dashboard/templates

Dashboard authentication properties
##
The values for localhost and remotehost can be:
none: No authentication is required
deny: All connections denied
ldap: Authentication is done by logging into an LDAP server and
optionally validating group membership
##
dashboard.ldap.url
Specify the LDAP host port and optionally a search base
##(ldap://<host>:<port>[/<search base>])
dashboard.ldap.url.group
Specify the LDAP host port and optionally a search base
##(ldap://<host>:<port>[/<search base>])
dashboard.auth=true
dashboard.auth=localhost=none dashboard.auth.remote=deny
dashboard.auth.ldap.url=ldap://localhost:389/ou=users,ou=system
dashboard.auth.ldap.url.group=ldap://localhost:389/cn=group1,ou=groups,ou=system
Server API properties

api.on=true api.audit.on=false api.user.registry=serverapi
/registry.txt api.user.registry.encryption.on=false api.remote.on=true
api.remote.ssl.on=true api.remote.ssl.client.auth.on=true
api.remote.naming.port=1099
api.remote.server.ports=8700-8900 api.truststore=testserver.jks
api.truststore.type=jks {protect}-api.truststore.pass=server
REST API

api.rest.on=true api.rest.auth=false
api.rest.auth.realm=Tivoli Directory Integrator REST API
api.rest.jmsdriver.name=com.ibm.di.systemqueue.driver.ActiveMQ
api.rest.jmsdriver.queue.sender.persistiance=false
api.rest.jmsdriver.queue.sender.timeToLive=60000
api.rest.jmsdriver.param.jms.broker=vm://localhost?brokerConfig=xbean:etc/activemq.xml #
api.rest.jmsdriver.auth.username
api.rest.jmsdriver.auth.password
The properties determine the default bind address and the remote bind address
for the Server API.
* means bind to all network interfaces. The Remote Bind Address overrides
the Default one.
Only one IP address should be set. No hostnames are accepted.

```

```

Mind that the java.rmi.server.hostname property is set implicitly to
equal the Remote Bind Address property when used.
This will cause the client stubs to create sockets on the specified
Remote Bind Address.
com.ibm.di.default.bind.address=*
api.remote.bind.address=*
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=configs
Timeout in minutes for configuration locks. A value of 0 means no
timeout. api.config.lock.timeout=0
Timeout in minutes for loading a configuration.
api.config.load.timeout=2
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=
Specifies a list of Server notification types, which will be suppressed.
Notifications of suppressed types will not be propagated by the notifications
framework. ## The notification types in the list are separated by spaces.
Wildcards may be included.
Example:
api.notification.suppress=di.al.* di.ci.start
The above example will suppress all Assembly Line related notifications as well as
notifications for starting a configuration instance.
If the property is missing or is empty, no notifications will be
suppressed. api.notification.suppress=di.server.api.authenticate
di.server.api.authorize.*
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]".
To enable the built-in JAAS Authentication mechanism, set this property to "[jaas]".
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=
This property specifies whether LDAP Group authentication is turned on.

```

```

If it is set to 'true', the group membership of the authenticating user
will be resolved and will be taken into account during authorization.
If it is missing, the default value 'false' is used.
api.custom.authentication.ldap.groupsupport=false
Specifies the name of the attribute of a user in LDAP that contains
a list of the groups of which the user is a member.
It is taken into account only if 'api.custom.authentication.ldap.groupsupport'
is set to true. api.custom.authentication.ldap.usermodelmembershipattribute=
Specifies how groups are named in the membership attribute of a user.
For example, if the user's membership attribute contains values, which
correspond to the 'objectSID' attributes of groups, set this property to 'objectSID'.
If the user's membership attribute contains distinguished names of groups,
then set this property to 'dn'.
The property is required in case 'api.custom.authentication.ldap.groupsupport'
is set to true. api.custom.authentication.ldap.usermodelmembershipattributecontent=
Specifies the name of a group's attribute in LDAP, which corresponds to the
way the group is named in the TDI User Registry.
For example, if LDAP groups are addressed in the TDI registry by their
common name, then set this property to 'cn'.
If the User Registry contains the distinguished names of the groups,
then set this property to 'dn'. api.custom.authentication.ldap.groupnameattribute=
Represents the LDAP directory context, where groups will be searched.
It is required only when LDAP group support is enabled
api.custom.authentication.ldap.groupsearchbase=
Optional property, which represents a list of space-separated attribute names.
Specifies attributes which have non-string syntax.
api.custom.authentication.ldap.binaryattributes=
JAAS Authentication properties

java.security.auth.login.config=
Enabling/Disabling FIPS Mode in TDI

If the below property is set to true then TDI will be enforced to run
in FIPS Compliant Mode.
The default value is false, i.e. TDI will not run in FIPS Mode by default.
com.ibm.di.server.fipsmode.on=false
Specify the unique ID for the TDI Server

This property helps a client connecting to the TDI server to identify
different servers
running on the same IP and the same port in different time.
(Default is DEFAULT_ID) com.ibm.di.server.id=DEFAULT_ID
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/tivihelp/index.jsp?toc=
com.ibm.IBMCI.doc_7.1
/toc.xml com.ibm.di.helpHost=publib.boulder.ibm.com/infocenter/tivihelp
/v2r1/index.jsp?topic= ## 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=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.IBMQM
systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.JMSScriptDriver
systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.ActiveMQ
MQ JMS driver initialization properties
systemqueue.jmsdriver.param.jms.broker=<host>
systemqueue.jmsdriver.param.jms.serverChannel=<channel_name>
systemqueue.jmsdriver.param.jms.qManager=<queuemanager_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
ActiveMQ driver initialization properties
Specifies the location of the ActiveMQ initialization file.
This file is used to initialize ActiveMQ on TDI server startup.
systemqueue.jmsdriver.param.jms.broker=vm://localhost?brokerConfig=
xbean:etc/activemq.xml
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>

Logging settings

When false, all log calls made through the TDI Log class will be discarded.
com.ibm.logging.enabled=true

IBM JavaScript Engine settings

Set the type of platform - required by the IBM JS Engine when caching is used.
com.ibm.commons.platform=com.ibm.commons.platform.GenericPlatform
##
Set this property to a directory to enable auto dumps of assemblylines that fails
##
com.ibm.tdi.autodump.directory=<dump directory>
##
Server API client properties
api.client.ssl.custom.properties.on=true api.client.keystore=serverapi/
testadmin.jks {protect}-api.client.keystore.pass=administrator
api.client.keystore.type=jks {protect}-api.client.key.pass=administrator
api.client.truststore=serverapi/testadmin.jks {protect}-api.client.truststore.pass=administrator api.client.truststore.type=jks

```

---

## Appendix C. Monitoring with external tools

This is a "first step" into the integration between Tivoli Directory Integrator (TDI), IBM Tivoli Monitoring (ITM) and Tivoli Netcool/OMNIbus. It started as a proof-of-concept of this integration scenario. The TDI-ITM and TDI-OMNIbus integration capabilities shown in this document are fully supported solutions shipped with TDI. The solutions are shipped in the examples directory but they are fully supported.

JMX was chosen for communication between TDI and ITM, because TDI provides a ready-to-use JMX interface and thus no development was necessary on the TDI side.

For monitoring TDI via Tivoli Netcool/OMNIbus an AssemblyLine was developed in order to detect TDI events and send them to OMNIbus. The purpose of this chapter is to present how TDI can be monitored by:

- Tivoli Monitoring using the TDI JMX interface
- Tivoli Netcool/OMNIbus

Both integration scenarios are bundled as official TDI examples and can be found in *TDI\_install\_dir/examples/Tivoli\_Monitoring* directory.

ITM 6.2.0 and Tivoli Netcool/OMNIbus 7.2.1 were used for these examples.

Several software components were necessary in order to realize the experiments described here. Here is the list of these components, and the reference documentation used to realize their installation:

- ITM Agent Builder 6.2 - ITM Agent Builder 6.2 User's Guide
- ITM Tivoli Enterprise Portal - ITM Tivoli Enterprise Portal online documentation
- Tivoli Netcool/OMNIbus 7.2.1 - Tivoli Netcool/OMNIbus online documentation.

JMX is used for communication between TDI and Tivoli Monitoring. On the TDI side it is the JMX layer of the Server API that Tivoli Monitoring connects to.

For communication between TDI and Tivoli Netcool/OMNIbus two connectors are used. A Server Notifications Connector is used to receive a set of TDI Server Notifications, and an EIF Connector to send events to OMNIbus.

---

## Monitoring TDI with ITM

### Short presentation of the ITM architecture

At its core, the browser provided by ITM presents data that is gathered by agents.

ITM agents are characterized by the following definition:

"The agents (referred to as managed systems) are installed on the system or subsystem requiring data collection and monitoring. The agents are responsible for data gathering and distribution of attributes to the monitoring servers, including initiating the heartbeat status." (Extract from the ITM documentation)

There can be various kinds of agents: agents to monitor operating systems or specific applications, or specifically tuned agents (that is, using the Universal Agent interface). The following diagram, taken from the ITM documentation, describes both the architecture and the deployment process of agents:

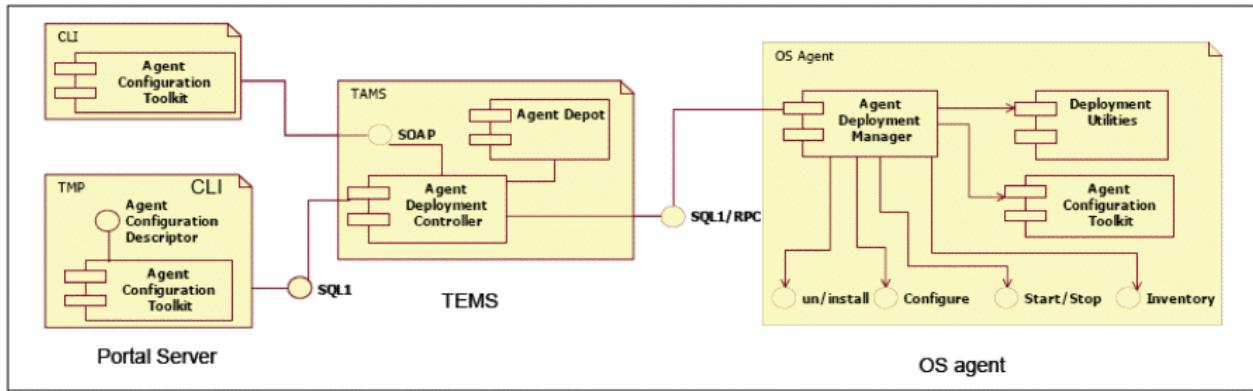


Figure 2. ITM Agents diagram

TEMS = Tivoli Enterprise Monitoring Services

TEP = Tivoli Enterprise portal

## Importing an existing Agent configuration in ITM Agent Builder 6.2

If you have an ITM Agent configuration XML file, you can import it in the ITM Agent Builder 6.2 and it will create ITM Agent project automatically. To import such a file, right-click in the ITM Agent Builder workspace, select **Import...** and select **IBM Tivoli Monitoring Agent for import**. Point to the configuration XML file (default name: `itm_toolkit_agent.xml`) and click **Finish**. This will create an ITM Agent Builder project with an appropriate name.

**Note:** If you wish to create the agent yourself, go to section “Creating a Tivoli Directory Integrator agent for ITM using ITM Agent Builder 6.2”; otherwise go to section “Generating the ITM Agent” on page 344.

## Creating a Tivoli Directory Integrator agent for ITM using ITM Agent Builder 6.2

The ITM Agent Builder is an Eclipse based platform for creating ITM Agents. The Agent that we will create for this example uses the JMX interface. From the ITM Agent Builder choose **File -> New -> IBM Tivoli Monitoring Agent**.

The ITM Agent Wizard will show up. The first step is an introduction – click **Next**. On the second step you will be asked to enter a project name. In this example we will use "TDI" as project name. Clicking **Next** brings us to the following step:

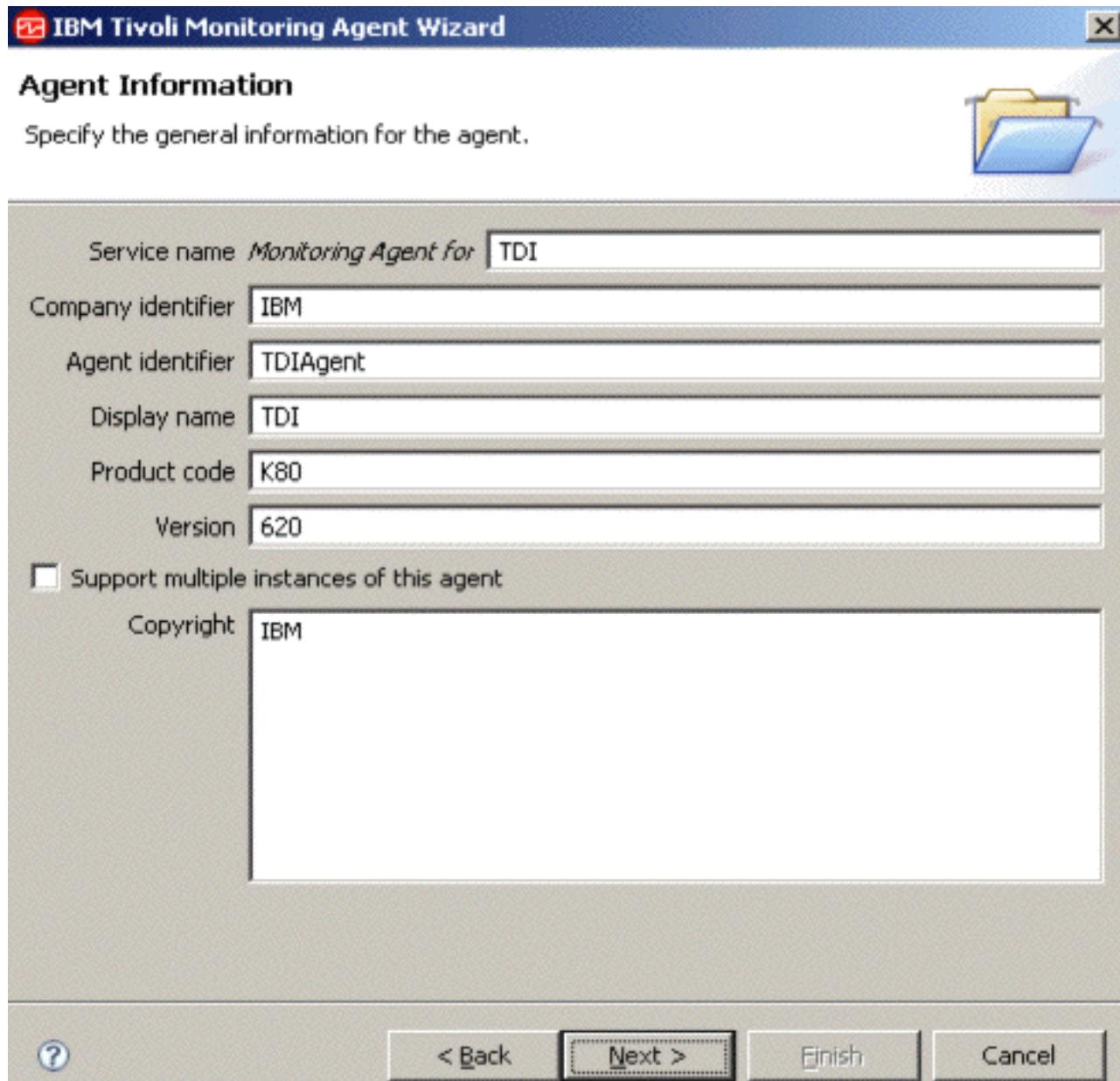


Figure 3. ITM Agent wizard Agent information

Fill all the fields with appropriate data. The Product code should be between K80 and K99 for JMX agents. click **Next**. On the next step check the **This agent will gather data from an external data source.** option and click **Next**. On this step the data source definition window is displayed:

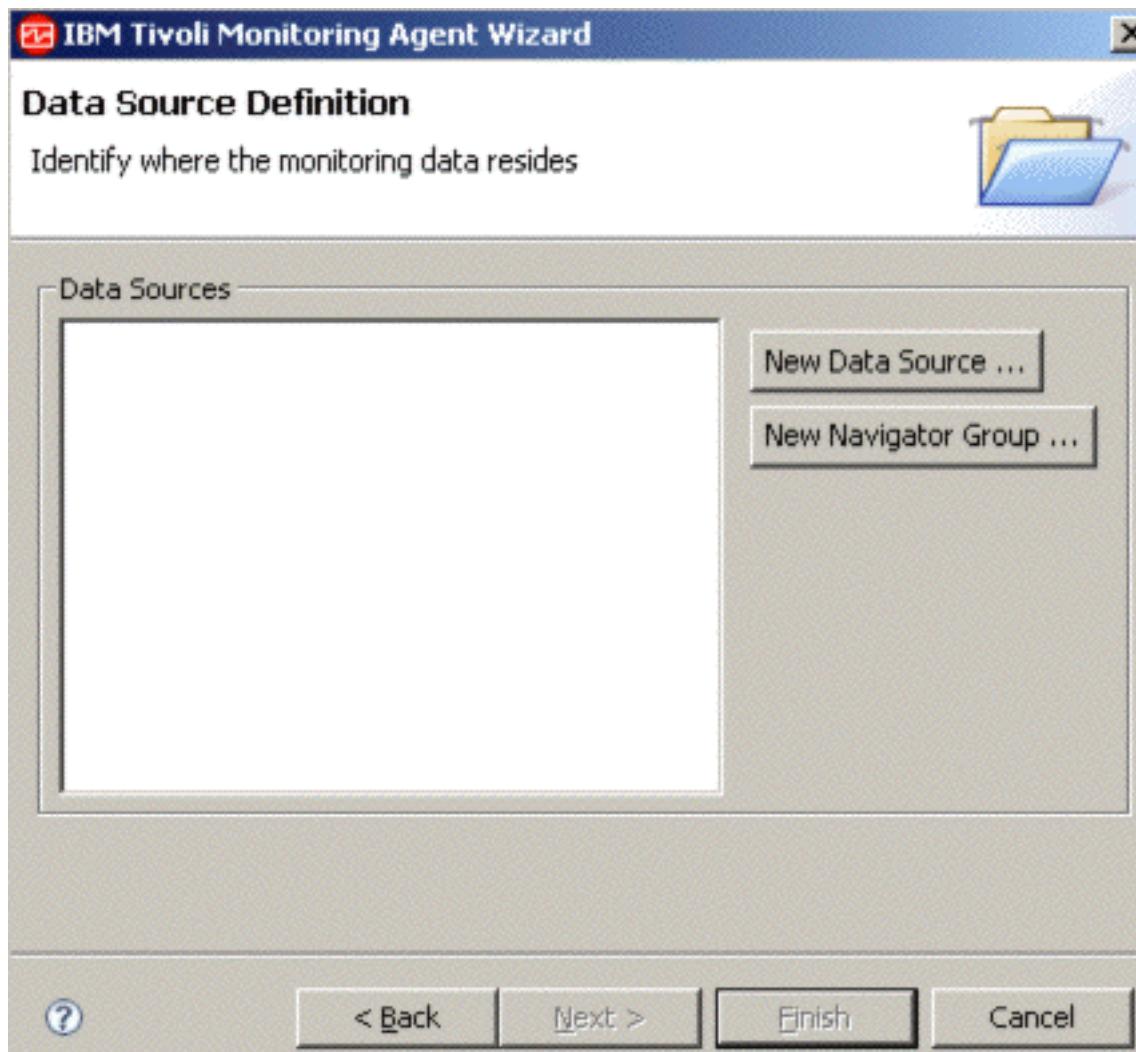


Figure 4. ITM Agent wizard, Data source definition

In order to make this step easier to configure start a TDI Server in daemon mode and run an AL that never ends (for example an AL with an HTTP Server Connector listening for connections). Make sure the JMX API is enabled in TDI (there is a description on how to do this later in the example).

Click the **New Data Source ...** button and then choose the **Collect data from Java Management Extensions (JMX) MBeans** option. Click **Next**. On the next window click **Browse** which should display the JMX Browser:

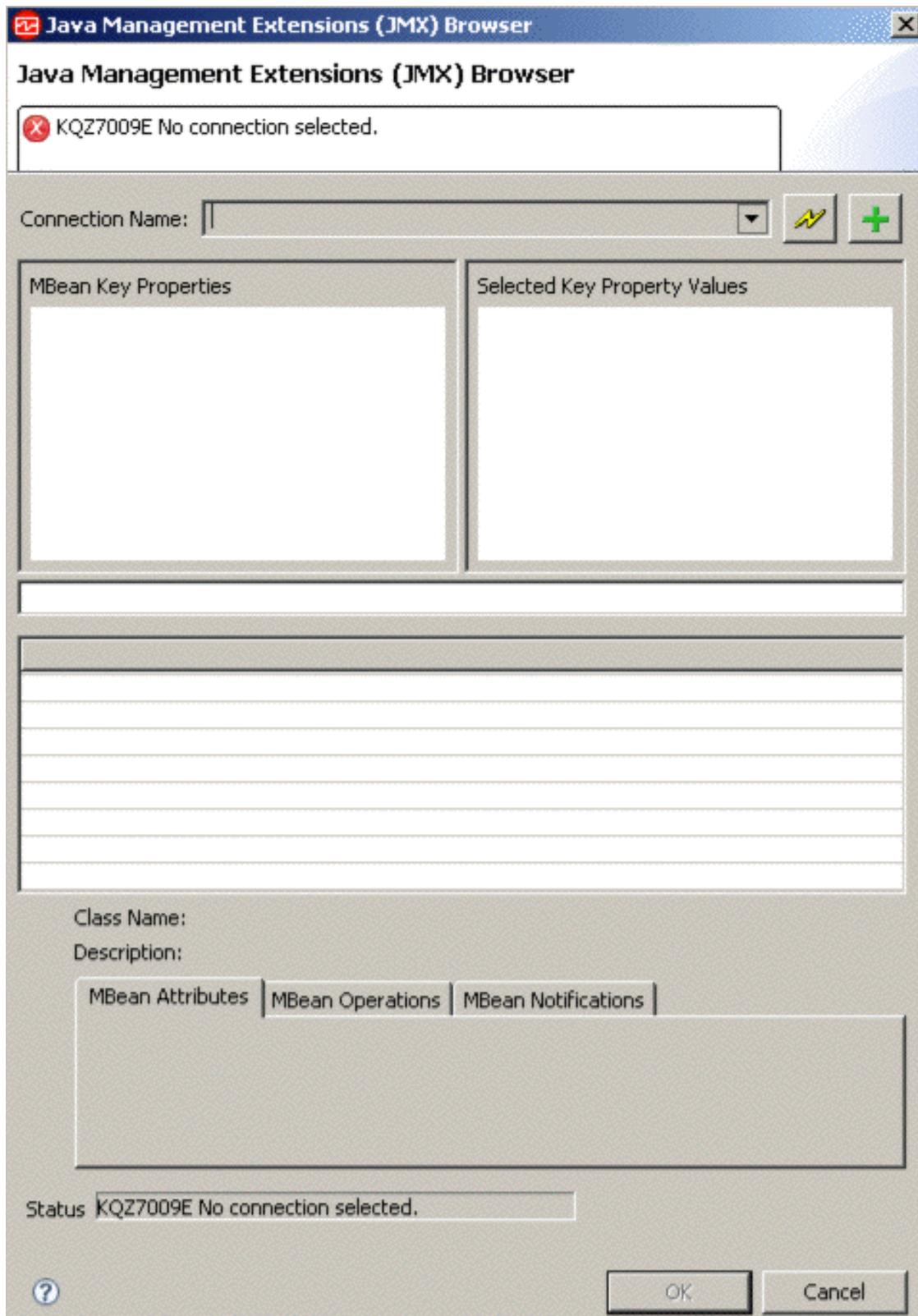


Figure 5. JMX Browser

Click the **Edit Connection Definitions** button (the green plus button). On the next step select **Standard JMX Connections (JSR-160)** and click **Next**. The new wizard window will display the available templates. Select **JSR-160 -Compliant Server** and again click **Next** to see the Connection properties of the

JMX Server.

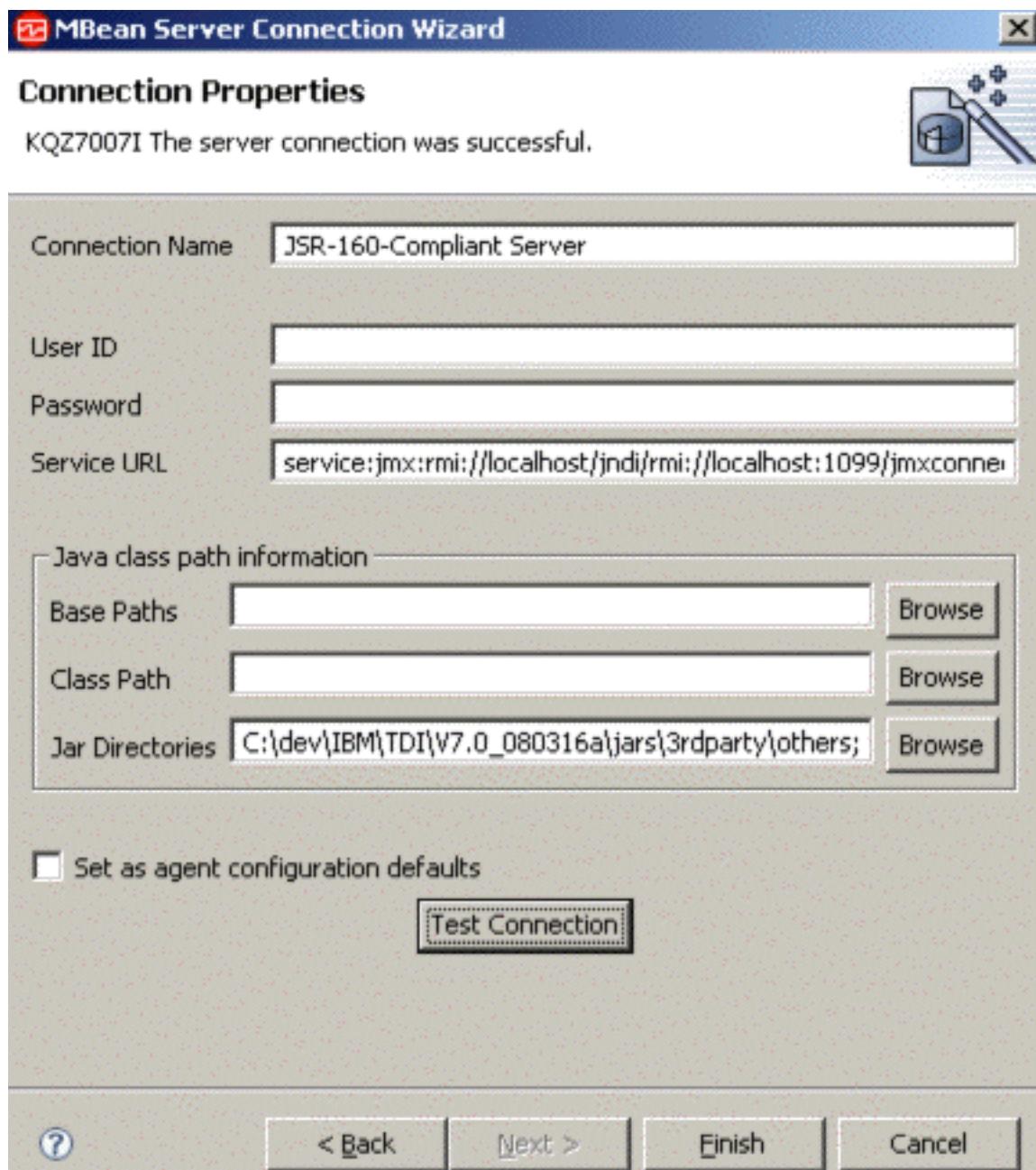


Figure 6. Server Connection wizard

In order to establish a successful connection with the TDI JMX Service you will need to enter a valid JMX Service URL (the default TDI JMX Service URL is `service:jmx:rmi://localhost/jndi/rmi://localhost:1099/jmxconnector`) and to configure the jar dependencies that are required for successful JMX MBeans creation (for the TDI JMX MBeans you will need the jar files in `TDI_install_dir\jars\3rdparty\IBM; TDI_install_dir\jars\3rdparty\others; TDI_install_dir\jars\common` directories). You can test these settings by clicking the **Test Connection** button. If the whole configuration is correct a message like this will be displayed: "The server connection was successful."

After this setup click **Finish**. The wizard should bring us the previous configuration step, but this time connected to the TDI JMX Server and will display additional information:

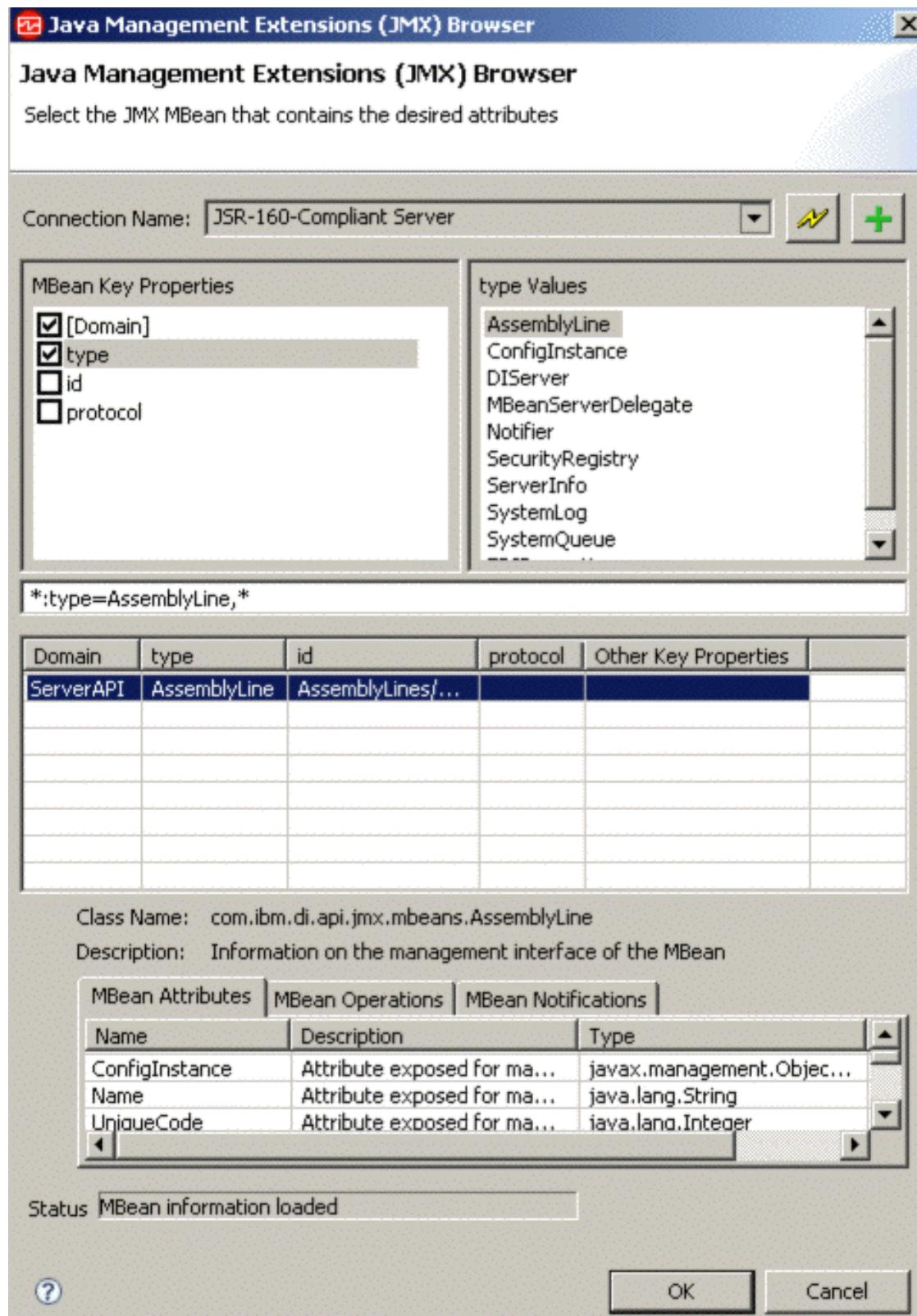


Figure 7. Browsing TDI in JMX Browser

Select the **type** MBean Key Property and **AssemblyLine** from the type values. To see the MBean Attributes you need to select a row in the table above them. In our case there is only one row. Click **OK** and then **Finish** to complete the setup of this data source.

Create one more data source with type value **ConfigInstance** in the same way we created the AssemblyLine data source. These two data sources will gather information from the JMX Server for running AssemblyLines and started Configuration Instances.

The third data source is a little different from the other two. It is a kind of listener which listens for notifications (events) sent by the TDI JMX Server. To create one like that, after clicking the **New Data Source...** button, you do not need to browse the JMX Server but simply enter `*:type=Notifier,*` for MBean pattern and click **Finish**. Two data sources will be created – one for the notification part and one for the static MBean part. Since we do not need the static part for this data source we need to remove it; right-click and select **Remove Data Source(s)**.

After completing these steps we should have three data sources created:

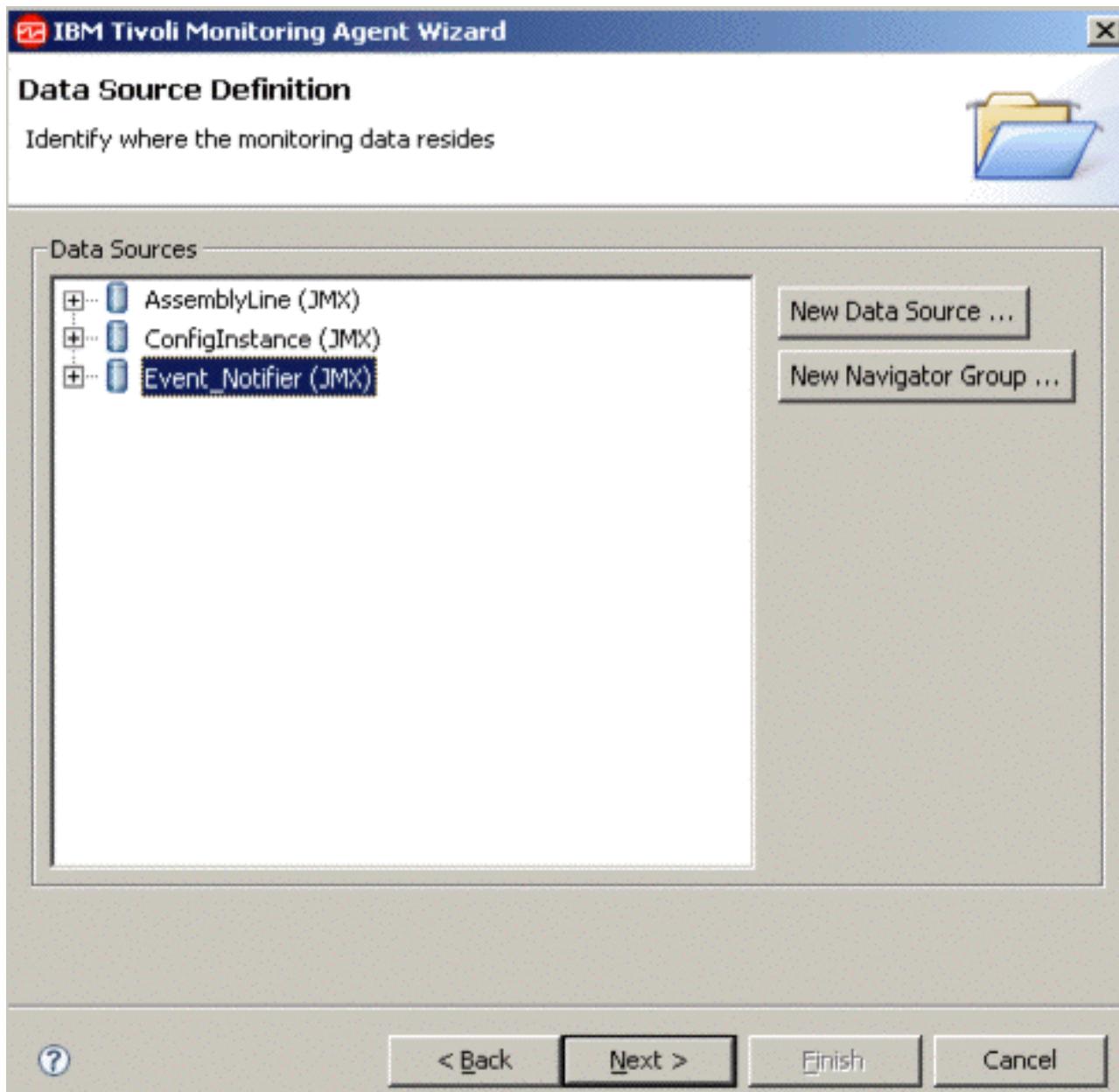


Figure 8. ITM Wizard, completed Data Source Definition

Expand the AssemblyLine data source and double-click the **ConfigInstance** attribute. In the ConfigInstance attribute configuration check the **key attribute** checkbox.

Expand the ConfigInstance data source and double-click the **ConfigId** attribute. In the ConfigId attribute configuration check the **key attribute** checkbox.

Click **Next** in order to configure the JMX Agent – Wide Options. Uncheck the JMX monitor attribute groups checkbox and select **JSR-160-Compliant Server** from the Server configuration choices.

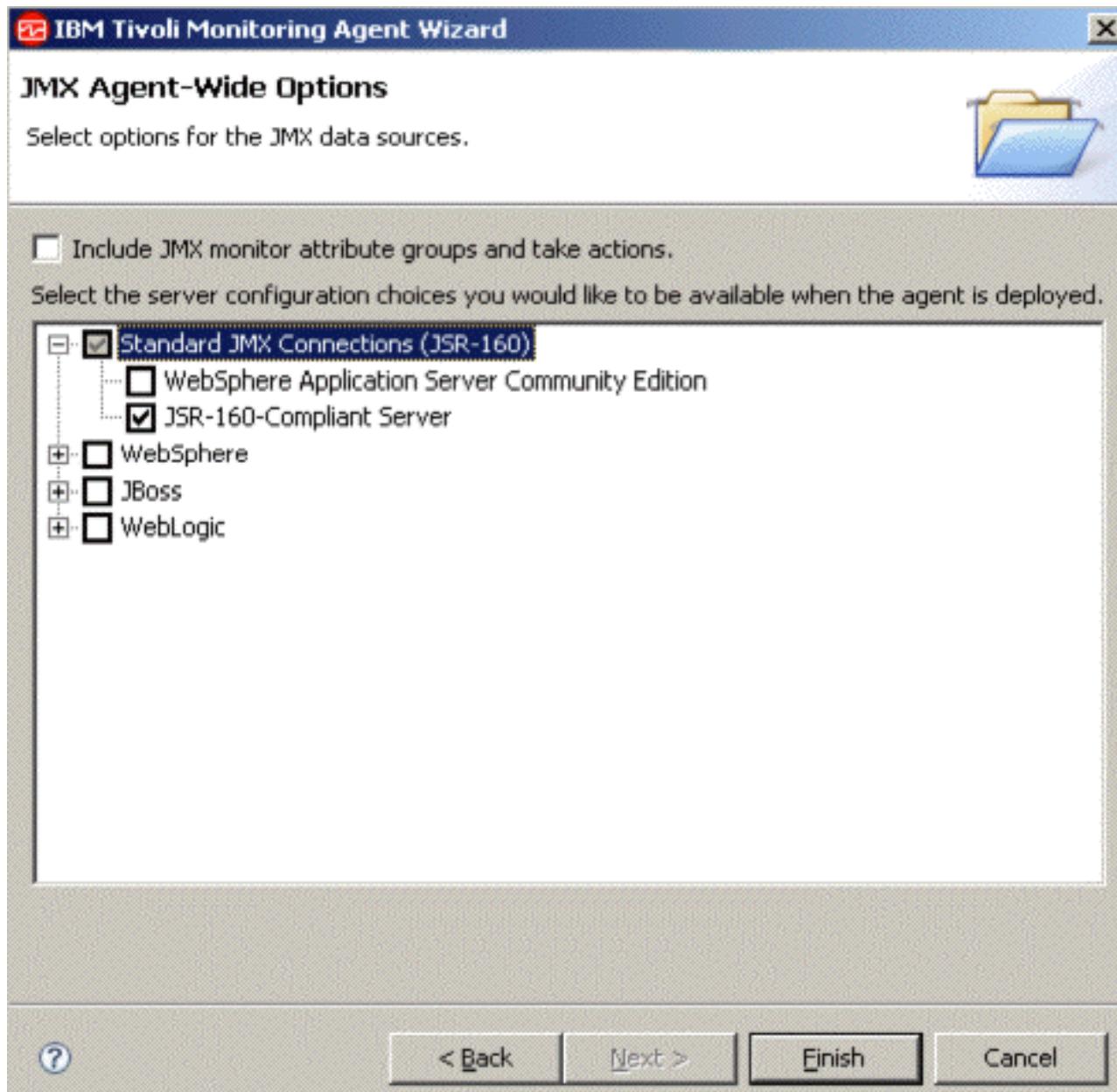


Figure 9. JMX Agent-wide options

Click **Finish** to complete the ITM Agent creation steps and save the Agent.

## Generating the ITM Agent

After the successful creation of the ITM Agent configuration, we need to generate it in order to deploy it in ITM. From the IBM Tivoli Monitoring Agent Editor menu in the ITM Agent Builder choose **Generate Agent**. The Generate Agent Wizard will appear. This wizard has several options of Agent generation. If you use ITM and ITM Agent Builder on a single machine then the **Generate the agent files in an ITM installation on this machine** option is suitable for you. The only field which needs to be configured is the ITM installation directory. Click **Finish** to generate and deploy the ITM Agent in ITM. This may take several minutes to complete.

**Note:** If you want to have the agent on another machine then you can use another agent generation option – **Create a compressed file so that the agent can be installed on another system**. This will

generate an archive that contains the ITM Agent installation. To install such an archived agent, you first have to copy the file to the machine where ITM is installed. Extract the files from the archive and from the command prompt start the `InstallIRA.bat` file with parameter the ITM install folder.

For example if ITM is installed in `C:\IBM\ITM` the command will look like:

```
<AgentDirectory>:\>InstallIRA.bat C:\IBM\ITM
```

## Configuring the ITM Agent

After the successful deployment of the agent (either using an archive file or the ITM Agent Builder option to deploy it on the same machine) we have to configure it in ITM. To do so, start **Manage Tivoli Monitoring Enterprise Services** where you can manage all ITM Agents:

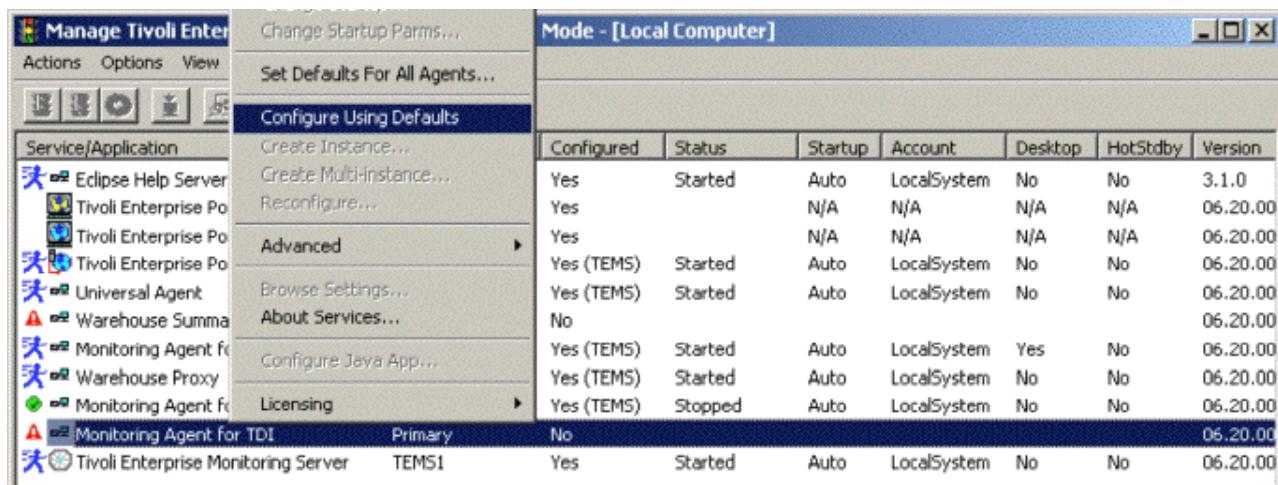


Figure 10. Manage Tivoli Monitoring Enterprise Services

Right-click the TDI Agent and select **Configure Using Defaults**.

On the next configuration window we need to configure the JVM properties for the Agent. Browse to the Java Home that you wish to use. The log trace level is set to "Error" by default. It can be changed to higher level in order to log additional information. After finishing the Java configuration click the **Next** button.

On the next configuration step you are asked to configure the JSR-160 Compliant Server properties, that is, enter username, password, Service URL and Class Path dependencies. For our example we need to enter Service URL and Jar directories like we did in order to create the agent – Service URL  
`service:jmx:rmi://localhost/jndi/rmi://localhost:1099/jmxconnector` and Jar directories  
`TDI_install_dir\jars\3rdparty\IBM; TDI_install_dir\jars\3rdparty\others; TDI_install_dir\jars\common.`

Click **OK** to complete the Agent Configuration.

The TDI Agent should be ready for use. The next step is to start the Agent. We can start it from the Manage TEMS window by right-clicking the TDI Agent and choosing **Start**. If all steps are successful the TDI Agent will be running now.

## Monitoring TDI data

To monitor data, we need to start the Tivoli Enterprise Portal (TEP), which is available in the ITM installation. In the navigator TEP window we can see the running Agents. The TDI Agent is also there and we have to expand it to see the specific monitoring data sources:

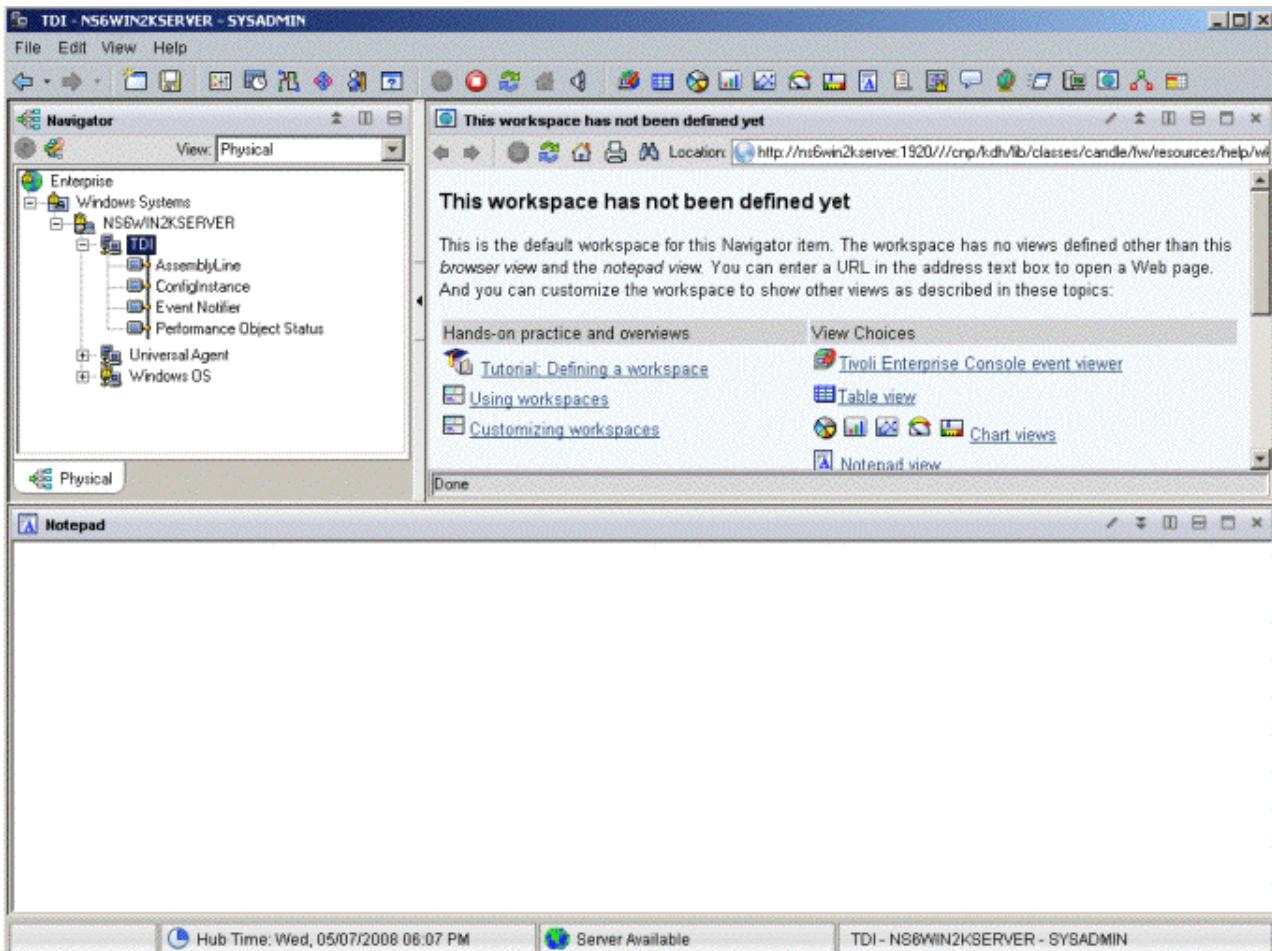


Figure 11. Tivoli Enterprise Portal (TEP) - Wizards

We can find our custom made data sources there – AssemblyLine, ConfigInstance, Event Notifier. If we have a running TDI Server and a running AL in it, we can see it in the report table of the AssemblyLine data source.

**Note:** In order to display data in the Notifier report table, the TDI Agent has to be running before a TDI notification is triggered.

This browser can be tuned in several ways: for numeric data it is possible to have a more human readable presentation (like diagrams). It is also possible to change the layout of the tables.

This functionality is not very complex, and is well described in the ITM documentation.

As the aim of this document is not to present the whole ITM product in intricate detail, but to focus on the usage that can be done in correlation with TDI, we will present here only the two trickiest concepts: defining thresholds, and links between tables.

For other functionality, please refer to the ITM documentation.

### Defining thresholds

To show how the threshold mechanism works, we will create the following simple example: display a warning when more than one AssemblyLine is currently running.

This threshold will depend on data provided by the AssemblyLine table. First we need to create a situation by right-clicking on the table of the agent, and by selecting **Situations** in the contextual menu:

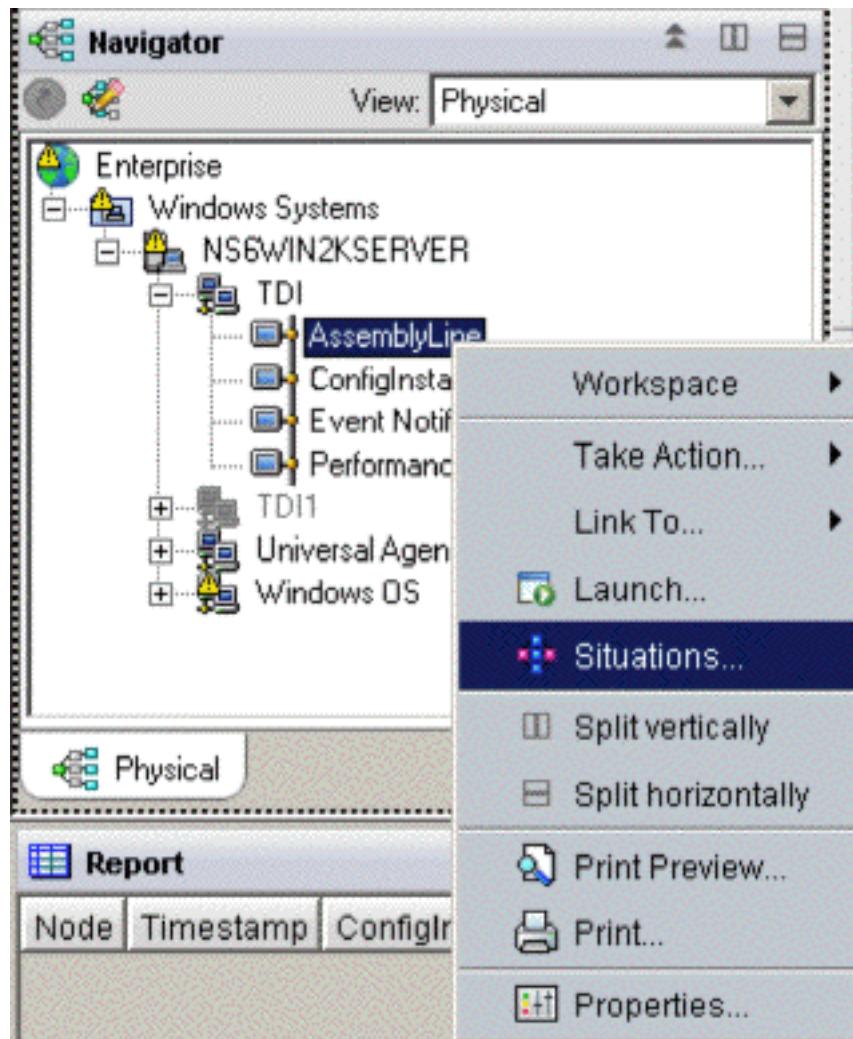


Figure 12. Situations context menu

Click on the **Create Situation** button in the upper-left corner and fill in the displayed form:

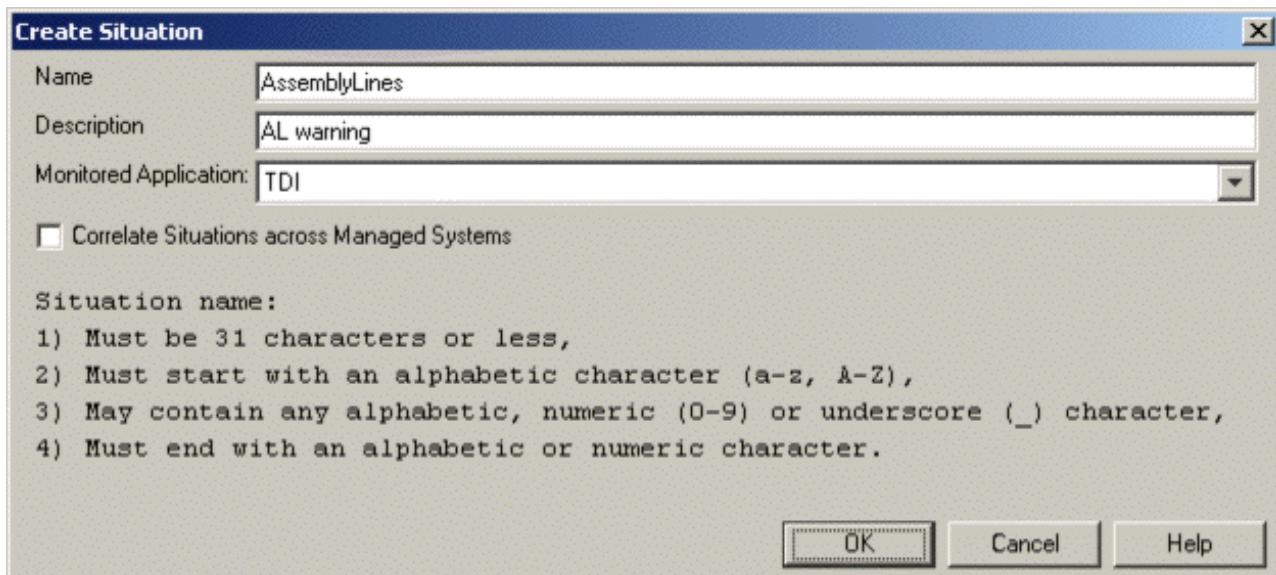


Figure 13. Situation form

This will be the name associated to the warning. Here we describe a case study, but in real situations you would give it meaningful names.

Then we choose with which table attribute our situation will deal:

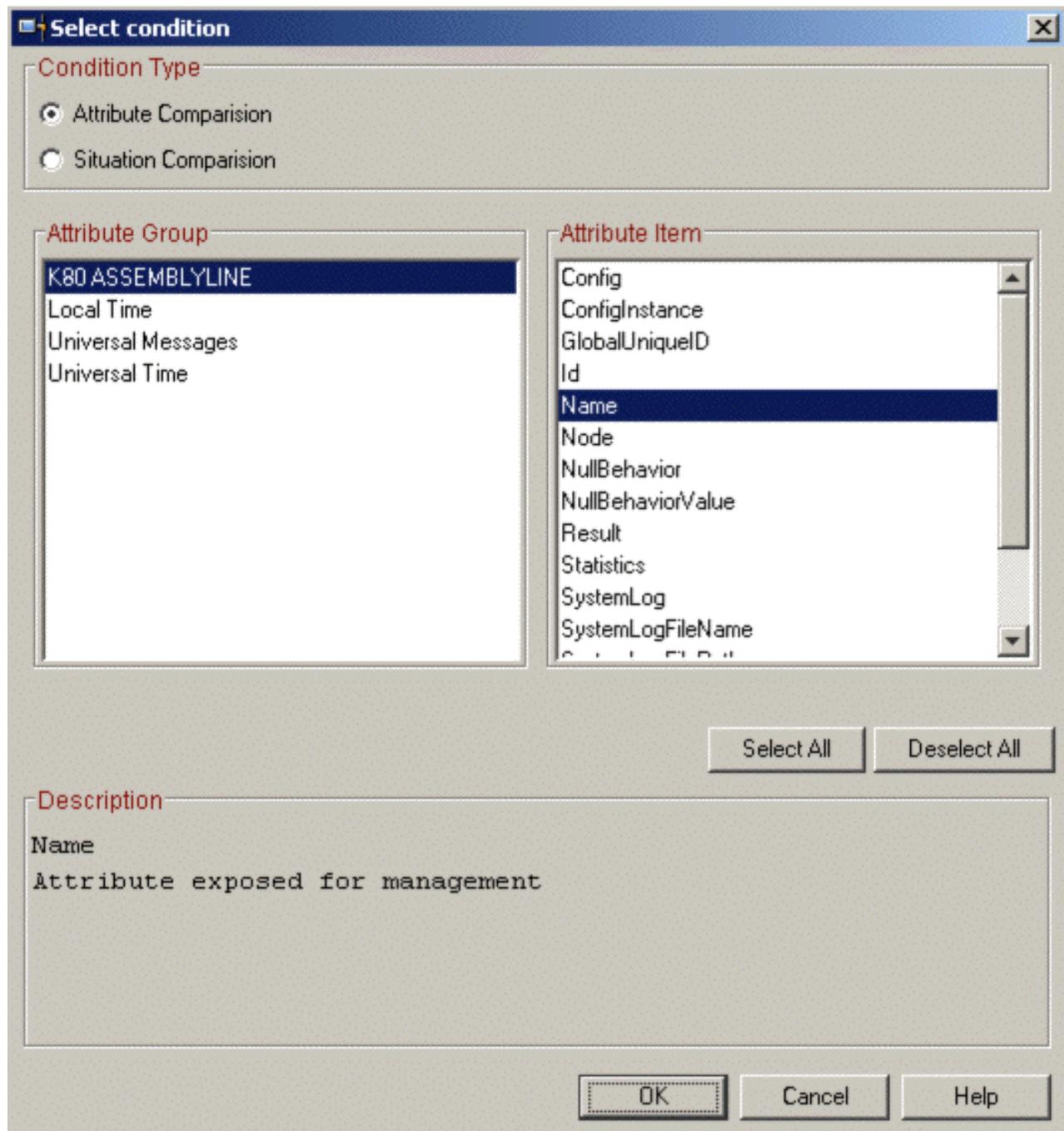


Figure 14. Situation: select condition

Indeed, we only have to consider the name of the AL to identify it.

Click in one of the cells, for example the one in line number 1:

| Name | <b>v</b> | <b>==</b> |      |
|------|----------|-----------|------|
|      |          |           | Name |
| 1    |          |           |      |
| 2    |          |           |      |
| 3    |          |           |      |

The display changes:

| Name | <b>v</b> | <b>==</b> |      |
|------|----------|-----------|------|
|      |          |           | Name |
| 1    | <b>v</b> | <b>==</b> |      |
| 2    |          |           |      |
| 3    |          |           |      |

Click on the **v**, and change it to "Count of group members".

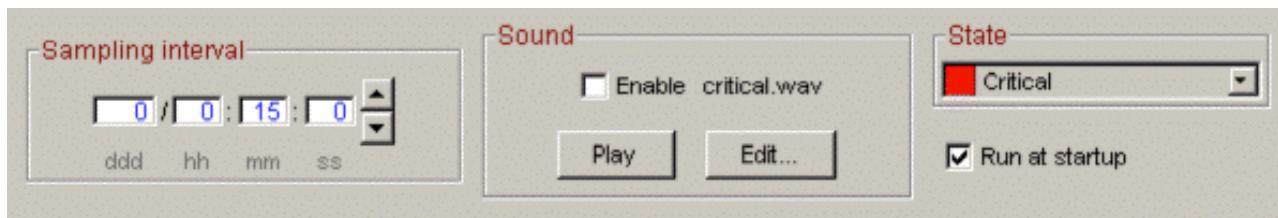
Click on the **==**, and change it to **>**.

Set the cell space remaining on the right to 1.

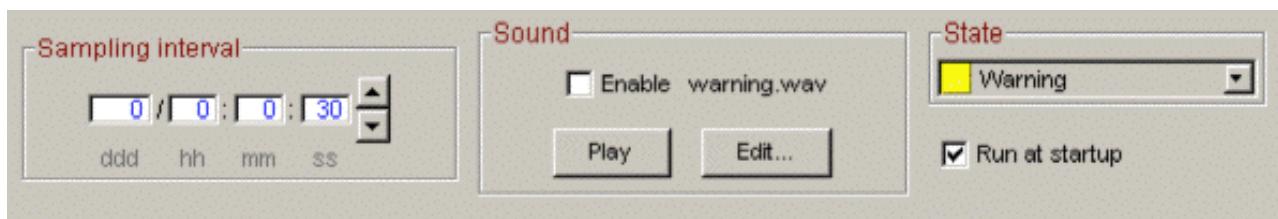
We have configured a condition to the **Name** column, which will be true if we have more than one AssemblyLine running:

| Name | <b>ML</b> | <b>&gt;</b> | 1    |
|------|-----------|-------------|------|
|      |           |             | Name |
| 1    | <b>ML</b> | <b>&gt;</b> | 1    |
| 2    |           |             |      |
| 3    |           |             |      |

And change the following default settings:



To this:



The situation is set, so **Apply** and validate this window.

Start Tivoli Directory Integrator Server and start at least two AssemblyLines at the same time; for example two HTTP Server Connectors that are listening on different ports.

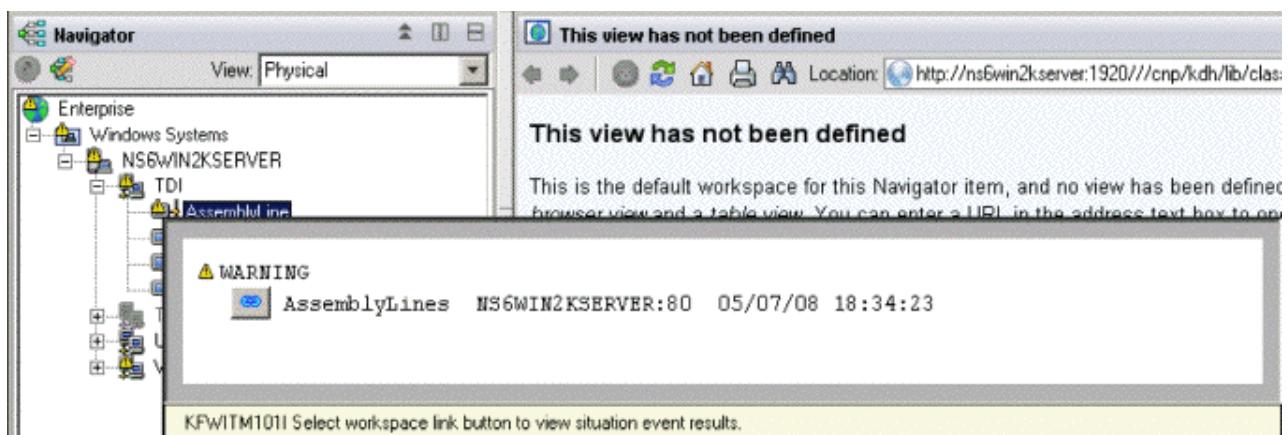


Figure 15. ITM displayed warning

The Warning window is opened while highlighting the warning icon.

### Creating links between tables

**Purpose of links:** It is possible to specify links between different tables in ITM. These links can be based upon some criteria of our choice, as we will see in this example. When a link is created you can automatically see a subset of a table by following the specified link. In this example we will create a link from the Event Notifier table that will show the currently running AssemblyLine in the AssemblyLine table. The link will be available in the Event Notifier table only for records that have the type "di.al.start". This type indicates that an AssemblyLine has been started. If the link is pressed and the AssemblyLine is still running, the AssemblyLine table will be automatically selected and only the corresponding AssemblyLine will be displayed in the table. If the AssemblyLine has already finished its execution then the displayed table will be empty.

This is an example Event Notifier table with defined link to the AssemblyLine table:

| Node              | Timestamp         | Type        | Source                              | Sequence Number | Time Stamp    | Message                                        |
|-------------------|-------------------|-------------|-------------------------------------|-----------------|---------------|------------------------------------------------|
| NS6WIN2KSERVER.80 | 05/07/08 18:33:59 | di.al.stop  | ServerAPI:type=Notifier,id=Notifier | 6               | 1210174439618 | AssemblyLine 'AssemblyLines/Serv               |
| NS6WIN2KSERVER.80 | 05/07/08 18:33:59 | di.al.start | ServerAPI:type=Notifier,id=Notifier | 5               | 1210174439558 | AssemblyLine 'AssemblyLines/Serv               |
| NS6WIN2KSERVER.80 | 05/07/08 18:33:59 | di.ci.start | ServerAPI:type=Notifier,id=Notifier | 4               | 1210174439558 | ConfigInstance 'runname' started.              |
| NS6WIN2KSERVER.80 | 05/07/08 18:32:49 | di.ci.start | ServerAPI:type=Notifier,id=Notifier | 3               | 1210174369672 | ConfigInstance 'sss' started.                  |
| NS6WIN2KSERVER.80 | 05/07/08 18:32:49 | di.ci.start | ServerAPI:type=Notifier,id=Notifier | 2               | 1210174369622 | ConfigInstance 'C_dev_IBM_TDI_V7.0_0' started. |
| NS6WIN2KSERVER.80 | 05/07/08 18:32:48 | di.al.start | ServerAPI:type=Notifier,id=Notifier | 1               | 1210174368120 | AssemblyLine 'AssemblyLines/Serv               |
| NS6WIN2KSERVER.80 | 05/07/08 18:32:36 | di.al.start | ServerAPI:type=Notifier,id=Notifier | 0               | 1210174355842 | AssemblyLine 'AssemblyLines/Serv               |

Figure 16. Example Event Notifier table

The table has three loaded configurations, three started AssemblyLines (one of which has been stopped). When the **ToTheRunningAssemblyLine** link is selected, the AssemblyLine is displayed in the AssemblyLine table (no other AssemblyLines are shown in the table):

| Name                  | Config  | ConfigInstance                                           | GlobalUniqueId | Id                      | Node             |
|-----------------------|---------|----------------------------------------------------------|----------------|-------------------------|------------------|
| AssemblyLines/Server1 | Server1 | ServerAPI:type=ConfigInstance,id=C_dev_IBM_TDI_V7.0_0... | 11210174355752 | AssemblyLines/Server1.1 | NS6WIN2KSERVER.8 |

Figure 17. Example Event

**Construction of links:** First we need to create a key in the AssemblyLine table which will be accessible from the Event Notifier table and will correspond to the AssemblyLine ID. Right-click in the AssemblyLine table and select properties.

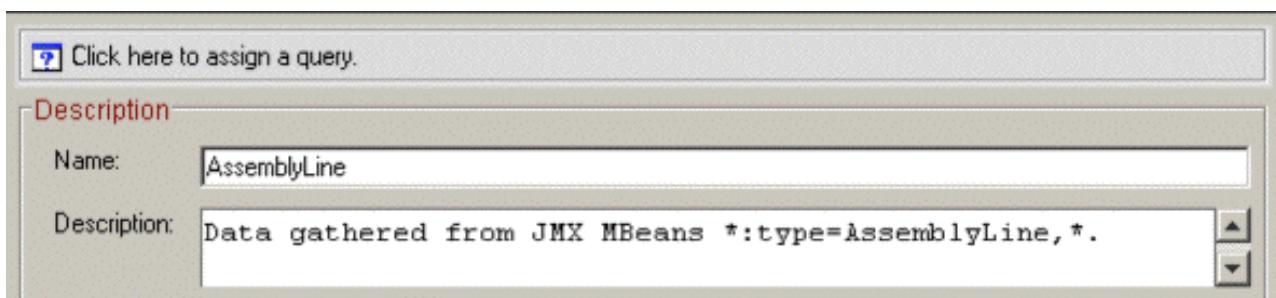


Figure 18. AssemblyLine properties

In the opened window, assign a new query by clicking the **Click here to assign a query** button.

The Query Editor will be opened where we must define another query because the existing one is static and cannot be modified. You will be asked to enter a name for it (for example "AssemblyLine2").

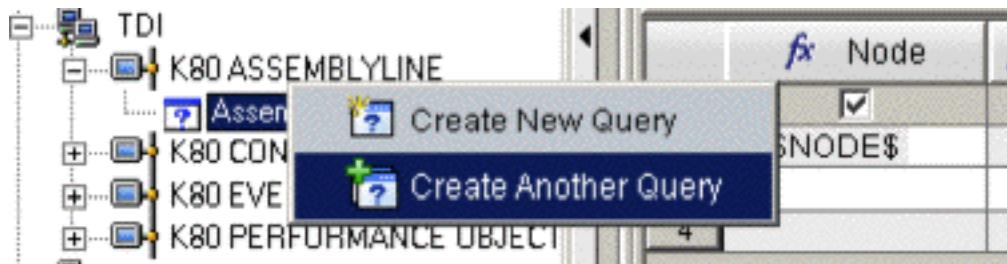


Figure 19. Create query selection

Go to the **Id** column of the table and enter \$keyid\$ as its value.

| Specification |                                     |           |                                     |                                     |                                     |
|---------------|-------------------------------------|-----------|-------------------------------------|-------------------------------------|-------------------------------------|
| <b>Id</b>     | <b>v</b>                            | <b>==</b> | \$keyid\$                           |                                     |                                     |
|               |                                     |           | <b>SystemLog</b>                    | <b>GlobalUniqueID</b>               | <b>Type</b>                         |
| 1             | <input checked="" type="checkbox"/> |           | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| 2             |                                     |           |                                     |                                     | <b>v</b>                            |
| 3             |                                     |           |                                     |                                     | <b>==</b>                           |
| 4             |                                     |           |                                     |                                     | \$keyid\$                           |

Figure 20. Query Editor

Click **OK** and apply the changes in the properties window and then click "OK" button.

This is all that needs to be done in the AssemblyLine table.

Go to the Event Notifier table (you will be asked to save the changed in the AssemblyLine table – click **Yes**).

Right-click on the selected row in the Event Notifier table and choose **Link To... -> Link Wizard...** (make sure that the type column of the selected row equals "di.al.start").

| Node              | Timestamp         | Type        | Source                              | Sequence Number | Time Stamp    | Message                           |
|-------------------|-------------------|-------------|-------------------------------------|-----------------|---------------|-----------------------------------|
| NS6WIN2KSERVER.80 | 05/07/08 18:33:59 | di.al.stop  | ServerAPI:type=Notifier,id=Notifier | 6               | 1210174439618 | AssemblyLine 'AssemblyLines/Serv  |
| NS6WIN2KSERVER.80 | 05/07/08 18:33:59 | di.al.start | ServerAPI:type=Notifier,id=Notifier | 5               | 1210174439558 | AssemblyLine 'AssemblyLines/Serv  |
| NS6WIN2KSERVER.80 | 05/07/08 18:33:59 | di.ci.start | ServerAPI:type=Notifier,id=Notifier | 4               | 1210174439558 | ConfigInstance 'runname' started. |
| NS6WIN2KSERVER.80 | 05/07/08 18:32:49 | di.ci.start | ServerAPI:type=Notifier,id=Notifier | 3               | 1210174369672 | ConfigInstance 'sss' started.     |
| NS6WIN2KSERVER.80 | 05/07/08 18:32:49 | di.ci.start | ServerAPI:type=Notifier,id=Notifier | 2               | 1210174369622 | ConfigInstance 'C__dev_IBM_TDI_V  |
| NS6WIN2KSERVER.80 | 05/07/08 18:32:48 | di.al.start | ServerAPI:type=Notifier,id=Notifier | 1               | 1210174368120 | AssemblyLine 'AssemblyLines/Serv  |
| NS6WIN2KSERVER.80 | 05/07/08 18:32:36 | di.al.start | ServerAPI:type=Notifier,id=Notifier | 0               | 1210174355842 | AssemblyLine 'AssemblyLines/Serv  |

Figure 21. Link Wizard selection

The Link Wizard will be displayed and will ask whether a new link will be created, modify existing link or delete existing link. Select **Create a new link** and click **Next**. On the next screen the name and the description of the link must be entered. In this example we will use **ToTheRunningAssemblyLine** as name and "Display the corresponding AssemblyLine in the AssemblyLine table." as description. The next step will ask to specify the link type. In the example we will use **Absolute** as link type because we are linking to a specified non-dynamic workspace in the navigator view. Proceed to the next step where we

must specify the workspace to which the link will direct (the AssemblyLine table).

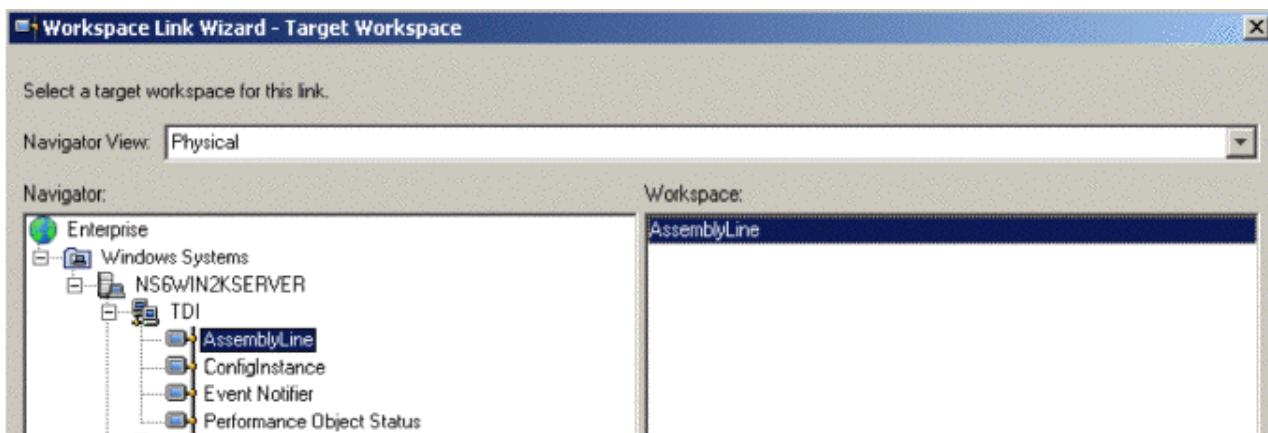


Figure 22. Link Wizard - target workspace

After selecting the AssemblyLine workspace click **Next**. This is the final step where we have to create the link conditions. We will modify two parameters in order to create the link properly – *contextIsAvailable* and *keyid*.

Select the **contextIsAvailable** parameter and click the **Modify Expression...** button (or double-click the parameter). The Expression Editor window will be displayed. Delete the current contents and click the **Symbol...** button. From the Symbols select the **Type** attribute:

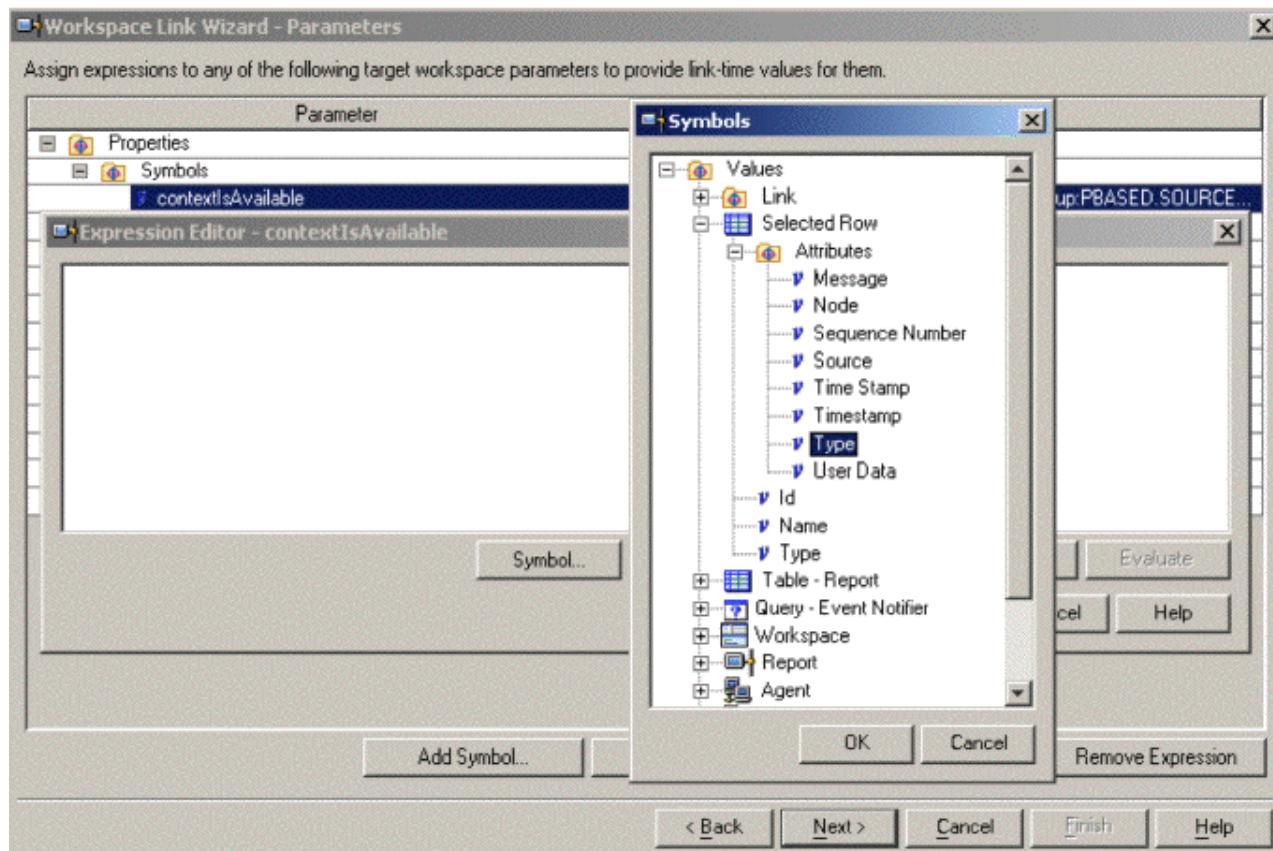


Figure 23. Link Wizard - Type attribute

Click **OK** to return to the Expression Editor window; add == "di.al.start" to create a conditional expression.

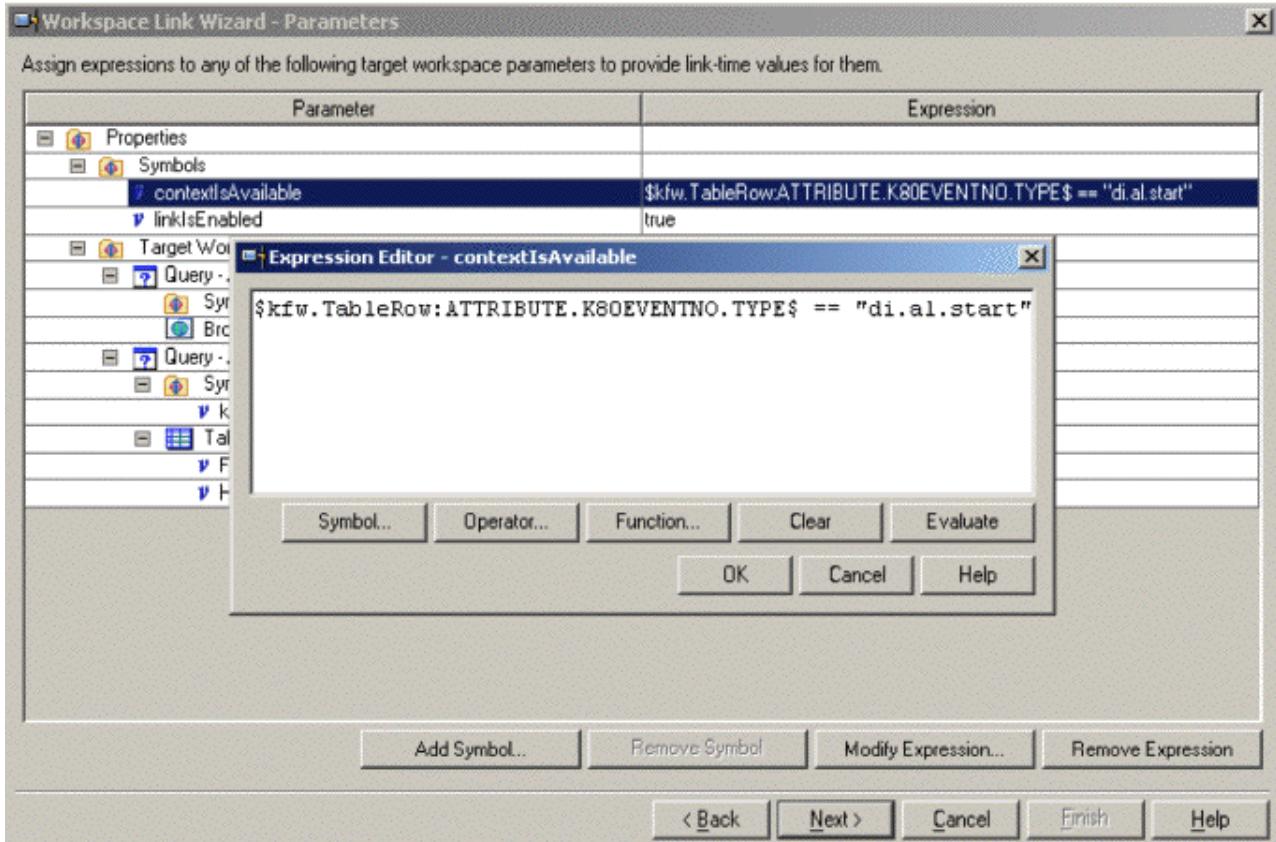


Figure 24. Link Wizard - Expression editor

Click **OK** to confirm the expression value.

Open the Expression Editor of the keyid symbol in **Query – AssemblyLine2** and add the User Data symbol.

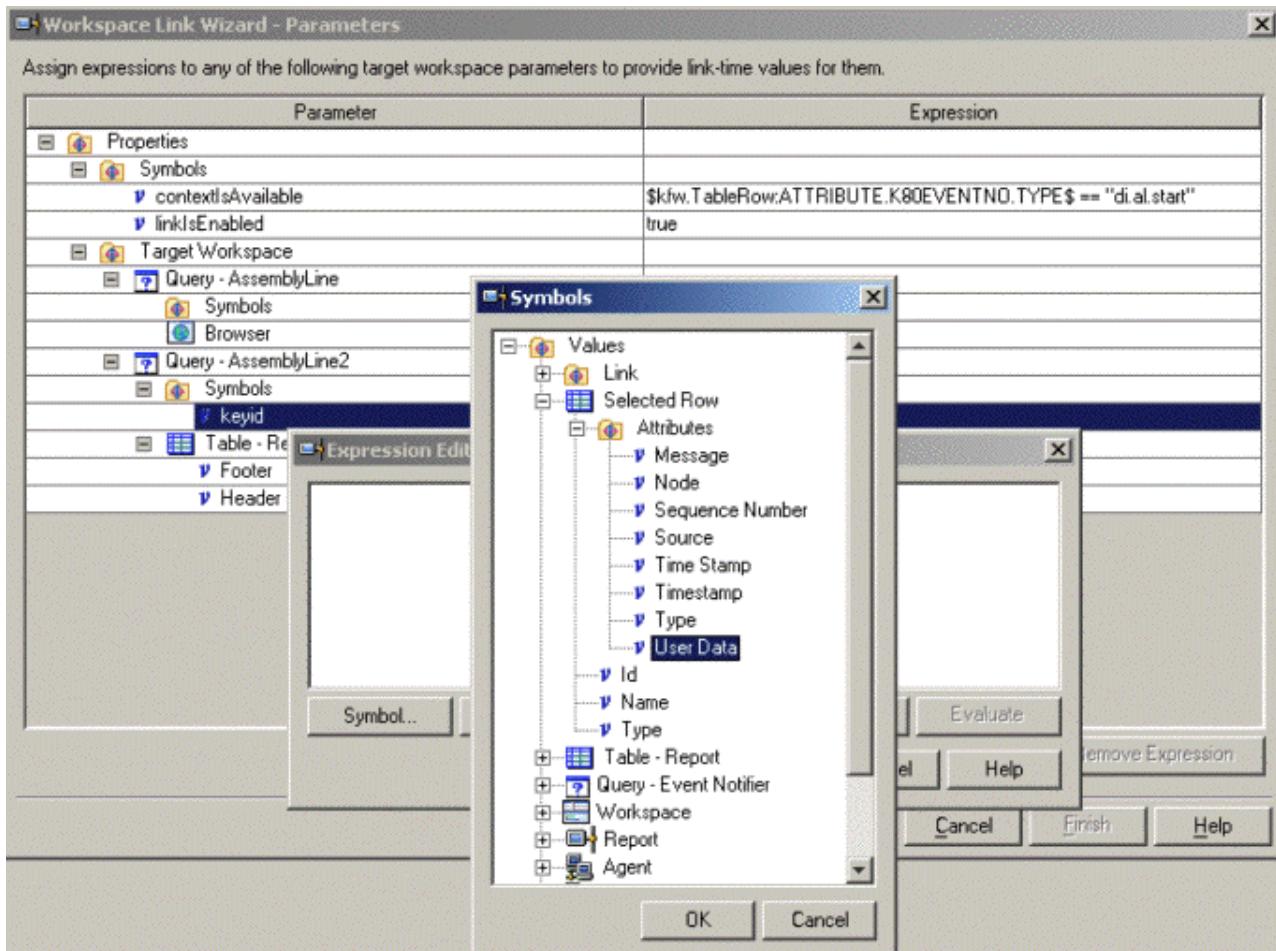


Figure 25. Link Wizard - User attribute

Click **OK** in the Expression Editor and click **Next** in the Link Wizard, which will show you a summary of the created link.

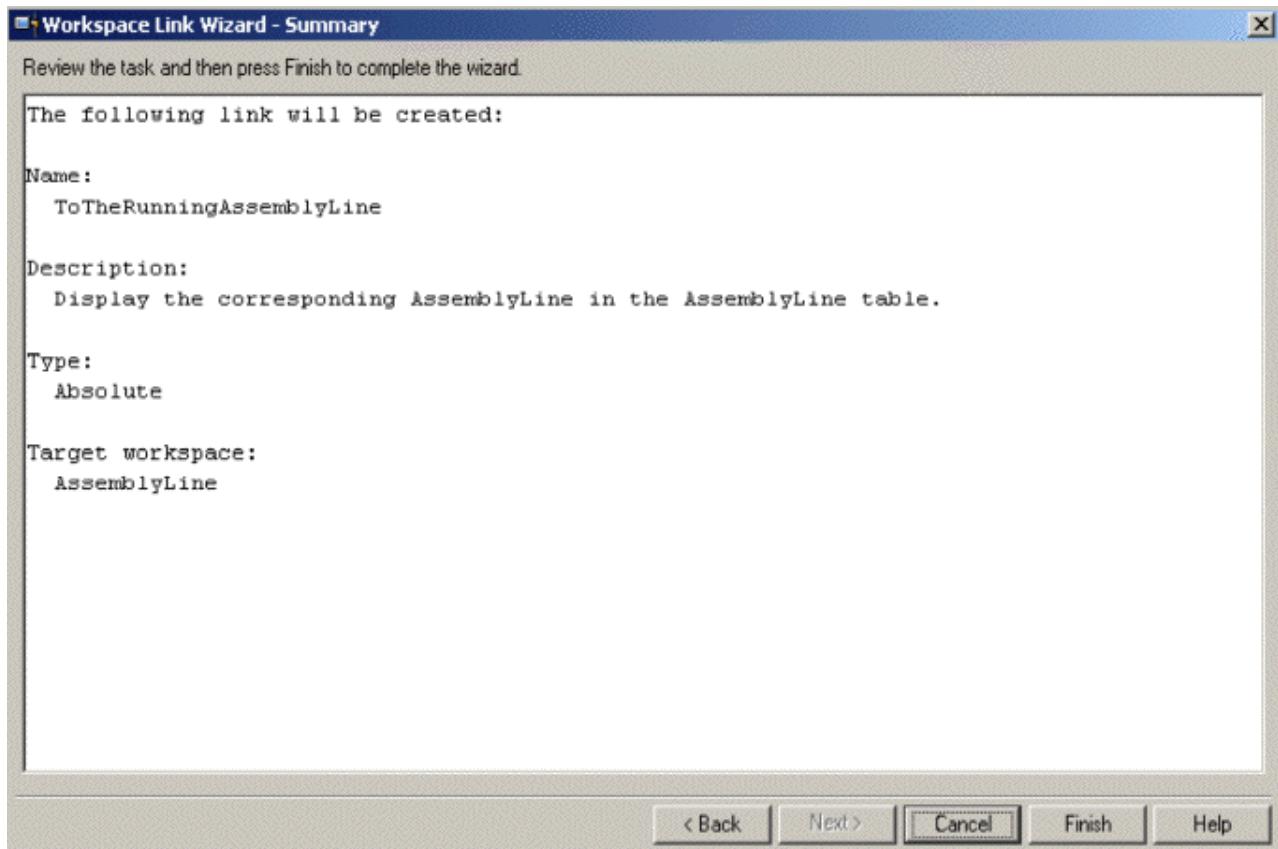


Figure 26. Link Wizard - summary

Click **Finish** to close the Link Wizard. Now you are ready to perform the steps in section “Purpose of links” on page 351.

## Send custom notifications to ITM

A configuration file that demonstrates sending custom notifications is shipped with the example. The file is located at *TDI\_install\_dir/examples/Tivoli\_Monitoring/TDI\_Monitored\_by\_ITM/custom\_notifications.xml*.

To send custom notifications you need to write your own script that does it. The following code demonstrates it:

```
session.sendCustomNotification(aType, aId, aData);
```

This piece of code sends a custom, user defined notification to all registered listeners. The **aType** parameter is the notification type. **aId** is the notifications ID. **aData** is custom user data. Note that the **aType** is automatically prefixed with "user.". This means that if you send a notification of type **myType** it will be received as **user.myType**.

## Limitations

The created Agent can not be used for managing the Tivoli Directory Integrator Server. For example it cannot start/stop AssemblyLines. It can be used for monitoring purposes only, even though the Tivoli Directory Integrator Server JMX layer exposes such methods.

## Monitoring TDI using OMNIbus

### Introduction

You can read more about OMNIbus in the section about the EIF Connector, in the *IBM Tivoli Directory Integrator V7.1.1 Reference Guide*.

### Configuring the EIF probe props file

In order to make sure that the port on which the EIF probe listens is the one that you expect you can set it manually.

To do this, refer to `$OMNIHOME/probes/<arch>/tivoli_eif.props` and set the value of property **PortNumber** to the number of the port on which the EIF probe will listen on.

By default if the EIF probe stays inactive (doesn't receive events) for more than 600 seconds, the service stops. You can set the timeout to infinity by setting the value of the **Inactivity** property to 0.

Your EIF prop file should look like this:

```
BufferEvents : "YES"
HandleMalformedAlarms : "true"
EIFCacheFile : '$OMNIHOME/var/tivoli_eif.cache' (Unix)
EIFCacheFile : '%OMNIHOME%\var\tivoli_eif.cache' (Windows)
EventCopies : 1
Inactivity : 0
MaxEventQueueSize : 10000
PortMapper : "false"
PortMapperNumber : 100033057
PortNumber : 9998
Retry : "false"
StreamCapture : "false"
StreamCaptureFile : '$OMNIHOME/var/tivoli_eif.stream' (Unix)
StreamCaptureFile : '%OMNIHOME%\var\tivoli_eif.stream' (Windows)
```

However, if you decide not to modify the EIF probe props file, be aware that the default port that the EIF probe listens to events is 9999 (according to the OMNIbus documentation).

### Determine the severity for the events

To determine the severity for the events you will need to do a few modifications to the EIF rules file. Here is a brief description of how you can manage the severity. For this purpose we will define any **start** events as low severity and **stop** events as high severity. Then you can type the following code in the rules file:

```
if(regmatch($ClassName, "^.*\.start$"))
{
 @Severity = 0
}
if(regmatch($ClassName, "^.*\.stop$"))
{
 @Severity = 4
}
```

Note that custom notifications will have the default severity which is 1. This will result in the following:

| Netcool/OMNIbus Event List : Filter="All Events", View="Default" |             |                                                                                                                                         |        |
|------------------------------------------------------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------|--------|
| File                                                             | Edit        | View                                                                                                                                    | Alerts |
| Tools                                                            | Help        |                                                                                                                                         |        |
| All Events                                                       | Default     |                                                                                                                                         |        |
| Node                                                             | Alert Group | Summary                                                                                                                                 |        |
| TDI                                                              | di.ci.start | [type=di.ci.start, id=Event_Sender_00762522abe6450c99b7d5912a4a778f, data=null, configid=Event_Sender_00762522abe6450c99b7d5912a4a778f] |        |
| TDI                                                              | user.t1     | [type=user.t1, id=333, data=Hello everyone, created=Wed Jun 24 08:32:00 PDT 2009]                                                       |        |
| TDI                                                              | user.t2     | [type=user.t2, id=222, data=Hello TDI, created=Wed Jun 24 08:32:00 PDT 2009]                                                            |        |
| TDI                                                              | di.al.start | [type=di.al.start, id=AssemblyLines/AL1, data=12, configid=Event_Sender_00762522abe6450c99b7d5912a4a778f]                               |        |
| TDI                                                              | di.al.stop  | [type=di.al.stop, id=AssemblyLines/AL1, data=12, configid=Event_Sender_00762522abe6450c99b7d5912a4a778f]                                |        |
| TDI                                                              | di.ci.stop  | [type=di.ci.stop, id=Event_Sender_00762522abe6450c99b7d5912a4a778f, data=null, configid=Event_Sender_00762522abe6450c99b7d5912a4a778f]  |        |

Figure 27. OMNIbus Event list

## Working with the EventPropertyFile.properties file

EventPropertyFile.properties provides a default set of events that can be received by the Server Notifications Connector. This will enable the user to configure the AL using the property file only. The property file has the following structure:

key=value

*key* determines the type of event and *value* determines if this event is received. Therefore mainly *true* and *false* values are used. However, the *event.customNotifications* key doesn't expect a Boolean value. It must be set the names of custom events that will be received. For more detailed information about custom notifications refer to Sending custom notifications to OMNIbus. A few other things should be considered as well in order to avoid any misunderstanding. To clarify, look at the following diagram showing the default set of events:

```

->event.all
 |->event.ci.all
 | |->event.ci.start
 | |->event.ci.stop
 |->event.ci.fileUpdated
 |->event.al.all
 | |->event.al.start
 | |->event.al.stop
 |->event.server.stop
->event.hasCustomNotifications
 |->event.customNotifications

```

As shown above some events include sub-events. If you enable an event then all sub-events will be received, no matter if they are set to *true* or *false*. This means that if you have

```
event.ci.all=true
event.ci.stop=false
```

then the event.ci.stop event will be received despite of it is set to *false*. In other words event.ci.all overrides its sub-events. However, if you have

```
event.ci.all=false
event.ci.stop=true
event.ci.start=false
```

then only the event.ci.stop event will be received.

By default the property file is set to provide all TDI Server notifications. If you want to modify this set of events you need to change the Boolean values to *true* (if you want the event to be received) and *false* (if you want the event not to be received). In other words, if you want to receive all events that notify about the start of some component then your property file should look like this:

| Name                         | Protected | Local Value | Server Value |
|------------------------------|-----------|-------------|--------------|
| event.al.all                 | false     | false       | false        |
| event.a.start                | false     | true        | true         |
| event.al.stop                | false     | false       | false        |
| event.all                    | false     | false       | false        |
| event.ci.all                 | false     | false       | false        |
| event.ci.fileUpdated         | false     | false       | false        |
| event.ci.start               | false     | true        | true         |
| event.ci.stop                | false     | false       | false        |
| event.customNotifications    | false     |             |              |
| event.hasCustomNotifications | false     | false       | false        |
| event.server.stop            | false     | false       | false        |

Figure 28. OMNIBus Properties

For working with the property file when considering receiving custom notifications refer to the section below, "Send custom notifications to OMNIBus."

## Send custom notifications to OMNIBus

To receive custom notifications you must set event.hasCustomNotofications to *true*. Then you need to specify the set of events that will be received. Note that all custom events sent by TDI are prefixed with "user.". This means that if you send a custom event of type *myType* then you must set:

```
event.customNotifications=user.myType
```

To specify more than one custom event you can use ";" to separate them. To clarify, imagine the following situation. You want to receive all custom notifications of type "user.myType1", "user.myType2", "user.myType3". Then your property file opened with a text editor will look like this:

```
##Determine if Server Shutdown events are received
event.server.stop=false
##Determine what Custom Notification events are received
##This property is used only if event.hasCustomNotifications is enabled
##Note that all custom notifications are prefixed with "user."
event.customNotifications=user.myType1;user.myType2;user.myType3
##Determine if Custom Notification events are received
event.hasCustomNotifications=true
```

In order to receive any types of custom events the `event.customNotifications` value need to be set to `"*"`. This will not specify the type of custom events that the Connector will listen to, therefore any custom event detected will be manipulated. The ITM example provides a configuration that can send custom notifications. It can be used to send custom notifications to OMNIbus, too. For more information about custom notifications refer to section “Send custom notifications to ITM” on page 357.



---

## Appendix D. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
1623-14, Shimotsuruma, Yamato-shi  
Kanagawa 242-8502 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 might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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  
2Z4A/101  
11400 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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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 in softcopy form, the photographs and color illustrations might not be displayed.

---

## Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Linux is a trademark of Linus Torvalds in the United States, other countries, or both."Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

Adobe, the Adobe logo, Acrobat, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

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

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



Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium and the Ultrium Logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

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





**IBM**<sup>®</sup>

Product Number: 5724-K74

Printed in USA

SC27-2705-01

