Tivoli software

IBM

Redpaper

Axel Buecker
Johan Varno

# Robust Integration with Tivoli Directory Integrator 7.0

## Contents

# Introduction

What is perfectly acceptable to one person is a potential disaster for another. The concept of *availability* is closely related to *risk tolerance*. This document illustrates the options and capabilities that an integration designer can use when planning, developing, and deploying an integration solution using IBM Tivoli® Directory Integrator.

The challenges discussed in this IBM® Redpaper™ publication are not unique to Tivoli Directory Integrator, nor are the approaches to mitigate them. However, Tivoli Directory Integrator is an unusually flexible product, and understanding its capabilities can significantly reduce the time and cost of developing a solution, and provide a better answer for the organization in terms of risk, performance, and scalability.

The audience that can benefit from this document ranges from the casual Tivoli Directory Integrator developer with simple integration requirements to the data center manager with 24x7 availability requirements on everything in the infrastructure.

The casual Tivoli Directory Integrator developer can find approaches in the document that, with little effort, can significantly increase the resiliency of the integrations. Similarly, advanced users and architects can recognize patterns and learn about Tivoli Directory Integrator services that enable the building of highly available, resilient, and scalable Tivoli Directory Integrator solutions.

# Potential sources of trouble are all around us

An IT infrastructure has a vast number of *moving parts* (parts that operate independently of each other). The software solutions that operate in an infrastructure cannot be aware of all of the parts and continue to function if one of those unrecognized parts fails. Certain organizations use monitoring software to track some of the moving parts. Middleware services, such as databases and application services, are mirrored and clustered to keep uptime to a maximum. Yet these countermeasures are costly and are therefore applied only to those parts of the infrastructure that are considered *critical* to business uptime. These services might not be available to someone designing an integration solution.

This section describes several potential sources of trouble. Whether the threats are mitigated depends on how important they are considered and to what length one is willing to go to automate operations, even when something happens.

## Network

Network problems of many kinds can, for a short or a long period of time, be indistinguishable from each other as seen from the systems in the network. Those problems, however, can typically result in a failure during a connection or a read, write, or delete operation.

Telecommunication outage can render a remote system inaccessible. Firewalls sometimes drop inactive sessions, requiring the endpoints to reconnect. Firewalls can use rules, which might be based on time or traffic, that can affect the protocols used, and source and target addresses. A solution that worked perfectly fine in the test lab can therefore behave erratically after being deployed. Routers, switches, and networking interface cards can fail. The failures are not always easy to detect; the problem might be small, such as a loose connector or a network component that has been replaced with something configured slightly different than what was there before.

### Data source or target

One of the systems (source or target) that is involved in the integration process can have its environment parameters changed. This step can affect the establishment of the connection, and can result in a failure during operations against the system.

Examples include changed, locked, or revoked credentials, changed indexes or schema in a database, and access rights to a directory, database, or file system.

### Runtime environment

The machine on which Tivoli Directory Integrator is deployed might experience resource constraints, such as disk full, no free memory (either for the operating system or the Java™ Runtime Environment), processor near or at capacity, or a hardware problem that ultimately can shut down the entire server. The source of the problem might not be obvious. It might be caused by the Tivoli Directory Integrator solution itself, or other services and applications on the machine. Without intending to do so, another administrator might have changed parameters that affect the runtime environment for Tivoli Directory Integrator.

The runtime category differs slightly from the others becauseTivoli Directory Integrator itself might not run anymore, and therefore is not able to determine what to do. Restarting the operating system or virtual machine might be necessary, requiring manual control or management services to perform this.

### Unexpected data

An obvious ability of integration services is to automatically read and write between diverse systems. However, one system might provide a text string, and another system expects this input as digits. The solution might not consider that the input data will ever contain anything but digits. A safe assumption is that one day it will. This is of course a trivial example, which quickly becomes complicated when you have to consider the requirements of the involved systems. The most common problem is *no data* where something was expected. What should one do when critical fields are blank? The real world has a way of getting *dirt* into the data. What was a perfectly well-performing *proof of concept* can be deployed to everyone's content, and perform without a problem until something unexpected occurs.

The options are to fix, ignore, fail, and report. Tivoli Directory Integrator has many built-in capabilities to help handle these situations, all starting with the Tivoli Directory Integrator developer understanding the nature of the real world.

## Handle trouble proactively

Tivoli Directory Integrator can be considered a general-purpose integration service; integrations can span a wide range of technologies and methodologies. Many options are available to the designer who wants to build resilience, scalability, and availability into the solution. However, the options depend on the context, and no single approach provides increased performance or availability for everyone. Therefore, discussing risk and the associated cost is necessary.

Trouble must be taken seriously and considered carefully, but does not always have to be mitigated. The point is that you must determine the level of acceptable risk versus the cost to reduce or remove it. Although complete automation might be the goal, it might not be a practical goal.

Several questions are helpful to start the risk analysis:

- Is 100% uptime really necessary?

  The quick reaction is *yes*, but that can cause a substantial cost to deal with all of the issues described in the previous section. A low-cost-looping batch file that catches exit errors can send an e-mail or invoke other actions to inform a system owner. If an organization is reasonably comfortable with a delay of updates in the range of minutes or hours, then establishing a 100% architecture perfection can be considered excessive.

- Is real-time data synchronization really necessary?

  Often, a solution is easier to build and maintain if a delay is acceptable, whether the delay is in seconds, minutes, or hours. Such solutions can be incorporated into existing job-scheduling services and run at regular intervals; the log or exit code status could be monitored in the same environment.

- Can failover be handled manually?

  Consider the reasons for a failover to occur. Almost anything but a server failure can very likely lead to similar failure in the failover environment. If that is the case, the situation can be handled much more effectively by alerting an administrator.

In later sections, we describe several approaches that illustrate how simple measures can make a big difference.

Tivoli Directory Integrator provides capabilities that enable developers to implement robust stand-alone integration solutions that might include the following solutions:

- A stand-alone Web based *Administration and Monitoring Console* (AMC) that provides monitoring of any number of deployed Tivoli Directory Integrator solutions. Custom events and reporting can be sent to the AMC from the deployed Tivoli Directory Integrator solutions.
- An ability to send events, by using loggers and protocols, to monitoring software that overlooks the health of the overall IT infrastructure.
- Automatic retries when there are transient connection problems with source and target systems.
- An ability to modify connection parameters at run-time so that an AssemblyLine can fail over to other Tivoli Directory Integrator systems.
- An execution pipeline where custom logic can be inserted (*hooks*) to handle any internal problem in the Tivoli Directory Integrator server.
- A command-line interface (CLI) that allows remote control and execution of integration jobs. A cron-job on one Linux® machine can control the execution of Tivoli Directory Integrator jobs on any other platform from Microsoft® Windows® to IBM z/OS®.
- A change detection engine that can keep track of processed records so that a restart can skip those records that have been committed to the target system (or systems).
- Interfaces with message queuing services that facilitate building robust solutions on one or multiple Tivoli Directory Integrator servers.

*Early planning is important.* A common source of problems is that solutions are mostly built incrementally, with little concern of deployment requirements that should have been considered at an early stage. A serial integration is by nature performance-choked by the weakest link. Similarly, a serial sequence is less able to utilize multiple CPUs and loosely coupled technologies that can significantly enhance performance and availability. Therefore, planning for these issues early in the development phase can ease maintenance at a later stage.

# Architectural styles for increased availability

Because of the architectural and programming model flexibility in Tivoli Directory Integrator, the deployment and technology permutations are so numerous that what matters to one Tivoli Directory Integrator user might be useless to another. “Overview of architecture options” briefly describes several approaches to increase the availability of a Tivoli Directory Integrator integration solution. In “Read from database and write a file report” on page 8 and “Microsoft Active Directory with Lotus Domino” on page 10, we add more details for the simple approaches.

## Overview of architecture options

As discussed previously, a minimum of evaluation should be performed before jumping to conclusions regarding high availability. The goal is *availability just above the level where it meets the business continuance requirements*. A company cares about 100% business continuity, which does not necessarily equate with 100% real time. Data propagation can often fulfill the business requirements with periodic and simple solutions. Obviously, the Customer Relationship Management (CRM) system must be running, but updating customer records based on orders from the ordering system might lag several minutes without creating any problem for the organization. Furthermore, the kind of data flow that Tivoli Directory Integrator performs can also affect the architectural options. A simple source-to-target solution is much easier to deal with than one in which data goes back and forth between multiple systems.

Given this background information, the remainder of this section illustrates, at a high level, various architectural alternatives for increased availability. Remember, no matter how the high level approach might look, unless you consider data error handling and other issues, solutions can fail, even before the availability measures are activated. See “Planning for availability: tips and tricks” on page 20 and “Proper Tivoli Directory Integrator design and scripting” on page 32.

The architectures described in this section are not mutually exclusive. Consider them designs that can be mixed as necessary. Failover is a subjective decision; certain systems must make the decision to failover, either by giving or taking control. The challenge is that assumptions can be wrong, and starting a backup service can result in two active systems at the same time. This approach works fine if the data flow logic can tolerate it, but it is something for you to consider.

### Batch file that keeps solutions running

*The concern is problems with data, or that the server systems fail.*

A simple batch file is often a good approach because it is simple to build and administer. For more detail, see “Automation with batch file” on page 26. The batch file approach can be used to start Tivoli Directory Integrator periodically using another scheduler system, or the batch file can itself execute Tivoli Directory Integrator in a loop, restarting the AssemblyLines continuously until a problem is encountered. This approach is considered a best practice when dealing with change detection against the source systems because it makes it easier to increase availability than if a single Tivoli Directory Integrator AssemblyLine ran forever, listening for changes on a source system. A scheduler can increase availability and maintain low costs. See “External job scheduler” on page 7.

### Availability through duplication

*The concern is Tivoli Directory Integrator server machine failure.*

There might be deployments where two identical instances of Tivoli Directory Integrator can perform the same work without regard to each other. If the number of changes occurring is low, the cost of writing them twice might be acceptable. That way, any one of the two Tivoli Directory Integrator systems can fail, but the other continues to work. Tivoli Directory Integrator provides a *Compute changes in target* capability that skips updating the target system if it already contains what Tivoli Directory Integrator intends to write into it.

### External job scheduler

*The concern is Tivoli Directory Integrator server machine failure or environment problems leading to access problems to the source and target systems.*

A job scheduler can invoke command-line calls and can typically react to failures and switch over to alternate calls. A job scheduler can execute the batch files (see “Batch file that keeps solutions running” on page 6), inspect the resulting error codes, and take appropriate action.

### Duplicated Tivoli Directory Integrator server

*The concern is Tivoli Directory Integrator server machine failure. Because the number of expected changes is high, only one Tivoli Directory Integrator server must be active at one time.*

Two identical Tivoli Directory Integrator solutions communicate using Tivoli Directory Integrator notifications. The one started as a secondary service does not cycle or execute its AssemblyLine as long as it keeps receiving notifications from the primary. After the primary stops sending events, the secondary server executes until the primary starts sending notifications again.

### Multi-stage data flows with high availability (HA) middleware

*The concern is how Tivoli Directory Integrator can handle high availability requirements for middleware services*

“Highly available queues and databases” on page 16 describes how persistent middleware can offer a high level of availability with little effort for our Tivoli Directory Integrator solutions. By using Tivoli Directory Integrator to push data to and pull data from these middleware services, HA is available without having to build it into Tivoli Directory Integrator solutions. Multiple Tivoli Directory Integrator servers can do the same job, yet only one record from the source is written to the destination. Servers can be brought offline for maintenance and updates without affecting the overall solution, except possibly reducing total throughput.

### Built-in entry-level monitoring and failover capabilities

*The concern is knowing if and when my Tivoli Directory Integrator solution or my server runs into availability problems.*

The Web-based Administration and Monitoring Console and Action Manager (described in “Understanding the Administration and Monitoring Console” on page 18) provide monitoring capability for Tivoli Directory Integrator solutions. They can also act as a watch-dog to take action when single AssemblyLines or servers fail. This capability can add a significant layer of security to any deployment, but does not replace enterprise-level monitoring systems as provided by IBM Tivoli.

### Using enterprise monitoring systems

*The concern is knowing if and when my Tivoli Directory Integrator solution or my server runs into availability problems, and knowing how to integrate my Tivoli Directory Integrator deployment into my enterprise monitoring environment.*

All of the previously described approaches can be used when integrating Tivoli Directory Integrator with monitoring systems such as IBM Tivoli Monitoring[1], IBM Tivoli Netcool/OMNIbus[2], or systems from other vendors. In “Monitor Tivoli Directory Integrator” on page 17, we describe how these systems can monitor the status and progress of Tivoli Directory Integrator and make decisions about when to start failover or backup solutions.

## Introduction to scenarios

In the next sections, we describe two examples of Tivoli Directory Integrator at work and how availability and performance requirements are affected in various ways. The scenarios do not describe how to build the solution themselves, but they describe typical issues that can affect the availability, and the options that can be applied to reduce the risk of failure.

Each scenario follows a theme, not limited to the specific scenario. By reading through these scenarios and the later sections about approaches, technologies, and services, you should be in a better position to understand the available options to design an appropriate solution that considers business risk, risk tolerance, development, deployment, and maintenance costs. The scenarios cover the following two topics:

- Read from database and write a file report
- Microsoft Active Directory with Lotus Domino

## Read from database and write a file report

Even a simple task such as extraction of a report from a database into a Microsoft Excel file can cause problems. In this scenario, we describe extraction from any source, such as from another file, a Web service, or LDAP directory. Other automated systems might depend on this file, so this file must be generated without any interruption of service.

A complete risk analysis requires accounting for practically every possibility. Although this scenario does not warrant such complex analysis, a simple *if-then* exercise is helpful. Make a list of the possible events that can happen and then describe what, if anything, can be done to mitigate the particular problem. This approach, see Table 1 on page 9 as an example, can also help others to better understand what business decisions you have made.

[1] More information about IBM Tivoli Monitoring can be found at: http://www.ibm.com/software/tivoli/products/monitor/

[2] More information about IBM Tivoli Netcool/OMNIbus can be found at: http://www.ibm.com/software/tivoli/products/netcool-omnibus/

*Table 1 Simple risk analysis*

| Problem | Reason | Solution |
|---|---|---|
| Cannot get to the database | A network problem exists between Tivoli Directory Integrator server and database. The database might be offline. | Because the problem might be transient, retry the operation several times, and then log the problem if it persists. |
| Cannot log on to the database | Credentials might have been changed. | The problem requires that you send an alert or e-mail to the administrator. |
| Error while reading from database | Database schema might have been changed, or an error exists in the vendor JDBC driver. | The problem might be permanent, based on changes in the database or other reasons. |
| Tivoli Directory Integrator solution fails | Scripting does not handle the exception, dirty data, or other unforeseen situation. | The choices always include: fix, ignore, or abort. If you expect this problem, write to a log file when error occurs, send email to a designated administrator, otherwise, stop the AssemblyLine from further execution. |
| Physical Tivoli Directory Integrator server fails | Everything from power to disk, memory failure, or similar problems occur. | Either ignore and accept the risk, or use the CLI from another machine. |
| Error while writing to the file | Possible reasons:<br>► The disk might be full.<br>► The administrator of the Tivoli Directory Integrator server has changed access rights.<br>► Credentials might have changed at the OS level. | Treat the same as “Cannot log on to the database” problem. |

*The simple approach* is to ignore all errors that occur in Tivoli Directory Integrator, and set a custom *exit code* to indicate abnormal error situations. Then, use a batch file to run Tivoli Directory Integrator, check for this exit code, and execute a command line e-mail if the appropriate exit code is detected. “Locating and adding new log files” on page 21 describes where logs can be found to uncover sources of the problem.

The AssemblyLine *On Error Hook* is the last line of defense that can always be called if a situation has occurred that Tivoli Directory Integrator cannot automatically handle. In “Minimum error handling in the Error Hook” on page 22 we describe simple steps that can reduce the forensics effort.

Adding a custom exit code is done with the following line of JavaScript:

```
main.shutdownServer(9);
```

This exit code is typically located as the last line of code in the *On Error Hook*, and then detected in a batch file that started Tivoli Directory Integrator in the first place. See “Automation with batch file” on page 26 for more information.

Numerous free or inexpensive simple tools are available to send an e-mail using the command line. Another popular approach is to use a command-line tool to send *Twitter* messages; many graphical Twitter clients that can capture and display the Twitter stream.

*The next level of defense* is to reduce the impact of transient network and connection problems. These can occur at any time and disappear without a trace. Therefore, a prudent approach is to retry the connection several times before shutting down an AssemblyLine and alerting an administrator. “Configure automatic reconnection” on page 24 describes this technique in more detail.

After you perform the steps, the only remaining problems, practically, are *invalid credentials*, *surprise database changes*, or *disk full*. You must address these problems manually, because no automatic capability can fix any of them (at least without resorting to substantial coding or configuration). In a more advanced scenario, you can change parameters of input or output, to read from a separate source or write to another disk. “Failover to another data source” on page 25 describes this approach in more detail.

## Microsoft Active Directory with Lotus Domino

The main theme for this scenario is detecting and handling changes in a source system. Tivoli Directory Integrator provides several features and services for this purpose to allow the developer an amount of flexibility regarding how to solve the challenge.

Although certain systems provide change information about their data, a simple file typically does not. The developer must determine the delta between two versions of the file. Tivoli Directory Integrator can help with this, which we describe later. However, Microsoft Active Directory (AD) provides an *event service* that Tivoli Directory Integrator can *hook* into. This event service can inform Tivoli Directory Integrator about any changes to the AD data, and Tivoli Directory Integrator can then retrieve the data for further processing. Contrary to some other LDAP system, AD does not provide information about *what* in particular has changed in the data record. Tivoli Directory Integrator can help with that too. See “Change detection” on page 30 for more details about the change-handling capabilities in Tivoli Directory Integrator.

### Availability approach

The simplest availability solution is based on the approach discussed in “Batch file that keeps solutions running” on page 6. Do not listen for AD change events, but run Tivoli Directory Integrator regularly from a batch file to read/poll the changes from AD and update the data in Lotus® Domino®. If a problem emerges, manual intervention is necessary. It might be the appropriate response to business requirements, risk, and investment willingness. Tivoli Directory Integrator updates the change pointer information for each record, so that it does not have to reprocess those records that were previously handled when Tivoli Directory Integrator starts again.

### Failover approach

The simplest approach for a failover environment is to have two Tivoli Directory Integrator servers running the exact same solution as described in “Duplicated Tivoli Directory Integrator server” on page 7. Minor modifications are necessary.

Both modifications are typically in place, even for a single Tivoli Directory Integrator server solution:

- Select the **compute changes** option on the Lotus Domino update Connector. This option keeps the two Tivoli Directory Integrator servers from rewriting the same data to Lotus Domino. Rewriting non-changed data might not be considered a problem, but various Lotus Domino users or applications have workflows that are triggered on changed data events, so this approach can remove possible confusion.
- If this solution also deletes records in Domino, the delete Connector must have its *on error* Connector enabled so that it does not fail trying to delete something that the other Tivoli Directory Integrator server already has deleted.

This failover approach has little resource impact on AD and Lotus Domino because the extra *read changes* against AD return a limited number of records. The more seldom Tivoli Directory Integrator solution is run, the more records are returned. Similarly, Lotus Domino is burdened only with an extra read operation for each record that is added, modified, or deleted.

For a surprising number of AD and Lotus Domino users, an acceptable approach is to regularly read everything in one of the systems and then look up each user in the target system to determine whether it exists there, whether the attributes need changing, or whether it should be deleted. The negligible performance impact of this approach is attractive to many customers, removing many possible sources of availability concerns.

## Make message queues and events work for you

Because building a quick Tivoli Directory Integrator solution to solve an integration challenge is simple, continuing to enhance early work to create the final solution is often tempting. This may be an appropriate choice, but a conscious effort should be made to evaluate the architecture for later availability and performance requirements. *Separation of duties* is a key term when performing this evaluation. Determine whether a single AssemblyLine should do all of the work, or whether the workload should be spread across multiple AssemblyLines, possibly across multiple servers.

Tivoli Directory Integrator provides mechanisms that can be used to achieve separation of duties. The appropriate choice depends on the factors that drive the need for separation:

- Performance

  Separating input and output processes is one way to handle data that is coming in faster than it can be written to the target in a single thread. By adding more writing processes, either on the same CPU or on other servers, a greater total throughput can be achieved against the target system.

- Availability

  By introducing a highly available message queue as a persistent intermediary storage, multiple Tivoli Directory Integrator services can drive data to it and consume from it. Servers can be removed from this architecture at run-time, only affecting total throughput as other servers pick up the data. The message queue buffers any data that exceeds the reduced throughput capacity. This approach results in a more available architecture for

unplanned problems, and it allows system maintenance on individual servers as the total system continues to function.

- Governance

  When multiple owners are involved and have various ideas about availability, security, and use of resources, insulation can be a helpful approach. Work can be done in stages and forwarded to other nodes for further processing as needed. For example, a Tivoli Directory Integrator server might be located on the inside of a firewall, collecting changes from the infrastructure, and then at intervals upload an encrypted file to a business partner, other business unit, or a cloud service. Then, another Tivoli Directory Integrator server might receive this information and perform the necessary changes under full control of the receiving party. A single Tivoli Directory Integrator server would have introduced all sorts of conflicts, for example, where it needs to be located, how it implements access control mechanisms, and more.

Rather than building a single AssemblyLine that does all the work, a helpful approach is to split the work across AssemblyLines. These do not have to run on separate servers; the techniques that are described here work equally well within a single server.

Tivoli Directory Integrator AssemblyLines can read and write to message queues, and send and receive events. The recipients of this data can be other AssemblyLines, but also other systems in the organization.

## A message queue is your best friend

The concept of *message queues* is somewhat like *e-mail for applications*. For many years message queues have been used for separation of duties, service based architectures, availability, reliability, and performance reasons. Many vendors provide message queue products that can scale for performance and availability.

The basic idea is that you send a message with data to a message queue system. The message queue system will then, depending on its capabilities and configuration, send the message to all participating systems that have requested copies. The system stores the data until a participant requests it. The system might distribute copies to all participants that have asked for it, or forward to other message queue servers that are connected. The message queue can deliver the message to many, or to a single recipient. The latter is perfect for scaling and availability purposes by ensuring that only the next available client receives the message, feeding potentially many clients with the messages for processing in a *when available* fashion. Although clients can go offline, the queue hands messages off to those clients that are still available.

Queues can be used between Tivoli Directory Integrator AssemblyLines and other systems; they can also be used only between AssemblyLines, whether on the same or separate Tivoli Directory Integrator servers.

An example deployment architecture is depicted in Figure 1 on page 13. From left to right, aTivoli Directory Integrator AssemblyLine reads and processes data from a data source and submits it into a message queue. Now this Tivoli Directory Integrator service can process the next data record, without being concerned about any subsequent processing.

The message queue sends the data to the next available Tivoli Directory Integrator AssemblyLine for further processing. By operating multiple AssemblyLines (be it on a single multi-core server or on multiple physical servers) the solution becomes scalable. The AssemblyLines that picks up the data package from the message queue processes the data further and delivers it to its ultimate target.

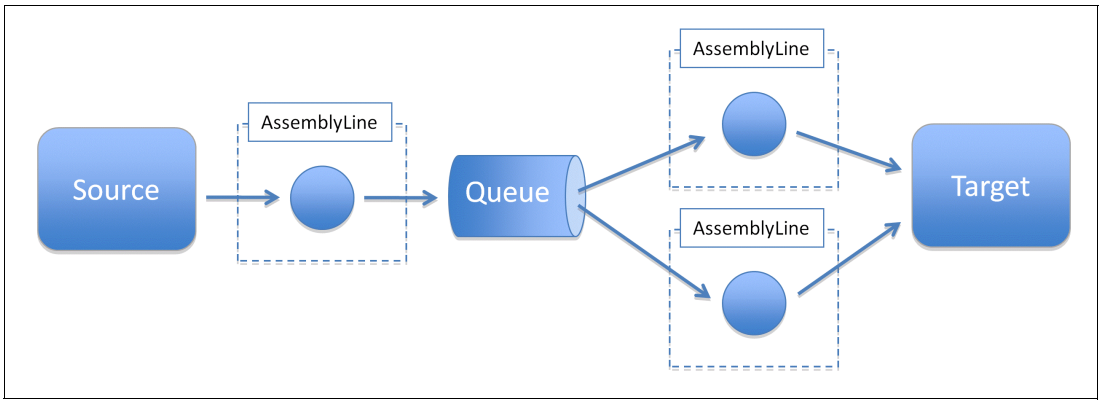


*Figure 1 Tivoli Directory Integrator deployment using message queue system*

This design is also appropriate for performance scaling when the bottleneck is the writing performance to the target. Often, such bottlenecks can be addressed with multiple threads or clients writing to the target, thereby achieving a higher total bandwidth of messages per second than is possible with a single client.

Many message queue products are available in the market. IBM pioneered this technology with its IBM MQ product that now powers mission-critical systems around the world. Tivoli Directory Integrator bundles a light-weight messaging system called IBM WebSphere® MQ Everyplace®, but supports nearly any system that supports the JMS standard.

## The Tivoli Directory Integrator SystemQueue

Tivoli Directory Integrator simplifies working with message queues by providing a SystemQueue abstraction layer that can interface with most message queue implementations. The SystemQueue simplifies passing data between Tivoli Directory Integrator AssemblyLines on one or multiple servers as it automatically serializes Tivoli Directory Integrator data and therefore removes all need of parsing. If you want to use message queues with systems other than Tivoli Directory Integrator systems, the JMS Connector allows you to use any parser to read and write data in the appropriate format.

As mentioned previously, Tivoli Directory Integrator includes a lightweight message service called MQ Everyplace that the SystemQueue connects to by default. Pointing it to another message queue is described in the SystemQueue[3] section of the *IBM Tivoli Directory Integrator Installation and Administrator Guide Version 7.0*, SC23-6560.

That guide also contains information about configuring MQ Everyplace (in the MQe Configuration Utility section).[4]

## Configure a JMS system for Tivoli Directory Integrator

Any JMS system can be configured to work with Tivoli Directory Integrator. The JMS Connector provides fields where the specific startup instructions for each system can be provided. In Figure 2 on page 14, the JMS Connector is configured with the Apache ActiveMQ[5] technology.

---

[3] Available at Tivoli Directory Integrator 7.0 Information Center Web site (System Queue): http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide48.htm#sysq

[4] Available at Tivoli Directory Integrator 7.0 Information Center Web site (MQe Configuration Utility): http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide50.htm#mqeconfutility

[5] The Apache ActiveMQ technology is freely available at http://activemq.apache.org/

In addition to the configuration of the JMS Connector, the Java JAR files must be copied to the following directory location:

```
<TDI install directory>/jars
```

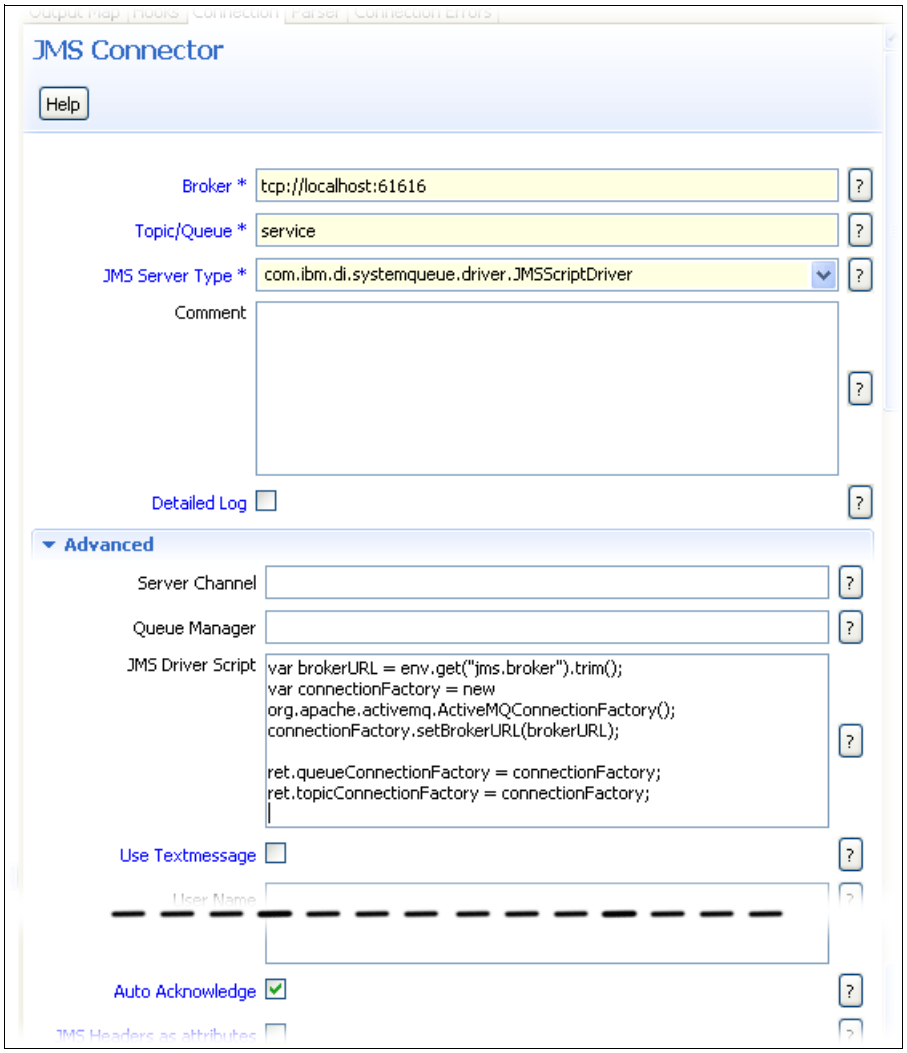


*Figure 2 JMS Connector configuration for Apache ActiveMQ*

## Using events to signal activity, status, or trouble

All network monitoring products support the SNMP protocol. It has been used for more than 20 years to poll and alert systems in the network. There are many other mechanisms available to request information and push events between systems. Monitoring systems can be set up to react to messages, but sometimes, even more importantly, they can react to a lack of messages, which might indicate that a system is no longer running or is temporarily inaccessible. By adding Connectors to an AssemblyLine that writes to log files or sends events, the appropriate decisions and actions can be taken at a higher level in the infrastructure. This can be other AssemblyLines that only wait for such events so that this behavior can be implemented independently of the main AssemblyLine that performs the main job.

Tivoli Directory Integrator can send and receive events by using various mechanisms. For simplification, sending and receiving will be treated separately. However, we first look at the built-in notifications inside Tivoli Directory Integrator that can be used for many purposes.

## Internal events: Server Notifications

A practical approach is to separate the main AssemblyLine that performs the main job and the one informing the environment about status, collecting audit information for log files, or handling problems that the main AssemblyLine can ignore. The internal Tivoli Directory Integrator events, called *Server Notifications,* can be used for this purpose. These Server Notifications allow the main AssemblyLine to send internal events for status and trouble, and have another service AssemblyLine (on the same or a separate server) receive the events and decide how to act on those events. That way, the service AssemblyLine can be taken down and modified at will without impacting the working AssemblyLine. A Tivoli Directory Integrator Event Connector can send and receive events within Tivoli Directory Integrator, even reaching out to other Tivoli Directory Integrator servers to receive events. More about Tivoli Directory Integrator notifications is in the "Server Notifications Connector[6]" section of the *IBM Tivoli Directory Integrator Reference Guide Version 7.0*, SC23-6562.

## Sending events

The Tivoli Directory Integrator Connectors can send events, such as the Tivoli Directory Integrator notifications (covered previously), SNMP, JMX, EIF, JMS, and e-mail, HTTP, and Web services. The event can anything, basically, that might represent an event if it is a common language between the sender and the receiver. Certain deployments might exists where Tivoli Directory Integrator invokes a command line, which again invokes a tool to alert an enterprise monitoring system.

Monitoring systems are capable of scanning log output and consider these as event sources. Tivoli Directory Integrator supports many types of loggers, including Microsoft Windows Events and the UNIX® syslogd, both commonly used for event handling.

Sending events is typically done from within an AssemblyLine. Messages can be sent for each cycle, providing an *I am alive* signal to other systems, but events can also be used less frequently when a potential problems exists, for example, when connectivity is faltering and the reconnect logic is activated.

## Receiving events

With the Tivoli Directory Integrator Iterator and Server modes, an AssemblyLine waits for incoming traffic. With the Server mode, the AssemblyLine can also respond to the event, which is appropriate when another system calls Tivoli Directory Integrator with protocols HTTP POST, REST, SNMP GET, or Web services call.

In its simplest form, an AssemblyLine might listen for Tivoli Directory Integrator server notifications of a certain type and write the information contained in the event into a log file.

The events can be used to modify behavior at run-time also. For example, a secondary AssemblyLine may monitor one of the protocols and turn on a debug flag that the main AssemblyLine can check for every cycle, and start logging more information until another event returns the debug flag to normal. Another use might be to signal a Tivoli Directory Integrator solution that it should fail over to another repository because the main one is down for maintenance. Similar to the other example, the main AssemblyLine checks on this every cycle and has script code to change the connection parameter, as described in "Failover to another data source" on page 25.

[6] Available at Tivoli Directory Integrator 7.0 Information Center Web site (Server Notifications Connector): http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/referenceguide47.htm#servernotconn

# Highly available queues and databases

You can reduce the complexity in the Tivoli Directory Integrator solution if you use capabilities in other systems. One of the greatest benefits of highly available middleware, such as databases and message queues, is that it can dramatically reduce the complexity of building availability into the other parts of the infrastructure.

Certain integration solutions can be made highly available by running two identical Tivoli Directory Integrator integrations on separate computers. This technique increases the load on the involved systems, but might be appropriate given the cost and risk trade-offs. However, when the Tivoli Directory Integrator solution becomes more complex there is a need for persistence, and that persistence also must be highly available.

The built-in Tivoli Directory Integrator services SystemStore and SystemQueue are by default configured to work with the Apache Derby[7] and MQe bundled products. These are powerful, yet lightweight products that in many scenarios perform their work excellently. Tivoli Directory Integrator has been designed to work with other products, and both the SystemStore and SystemQueue can be configured to work with other RDBMS and message queuing systems. By using a highly available IBM DB2® and IBM WebSphere MQ server, Tivoli Directory Integrator solutions can be developed that are highly resilient.

In a previous section, we used the expression *resilient* versus *available*. The reason is that with a highly available persistence system such as DB2 or MQ, the Tivoli Directory Integrator solution, if designed with a little care, can be deployed on a single server (if restarting the Tivoli Directory Integrator server manually or automatically when an error is detected is acceptable).

The next step is a fully redundant solution that keeps working no matter what happens. At this stage, you must have redundant Tivoli Directory Integrator servers also, and issues must be addressed, such as ensuring the removal of duplicates of the same message that might come into the integration solution through multiple active endpoints.

Although this section describes high availability, many scenarios can be made highly available without resorting to overly complex and costly high availability services.

To help you better understand and plan the use of message queues in the context of a highly available Tivoli Directory Integrator solution, we provide more details.

## Queues for availability

The design in Figure 3 on page 17 can also be implemented on only two servers running the identical integration solution, with AssemblyLine S1 and T1 on one machine, and S2 and T2 on another. The message queue in this case is implemented on both servers in a highly available configuration. Any one of the servers can fail, and the other is able to continue, although at half throughput speed. The failed server can be fixed and brought back online without taking the other system down. This is not a trivial deployment, but one that adds significant resilience to the integration architecture.

[7] For more information about the Apache Derby Java relational database management system, go to: http://db.apache.org/derby/

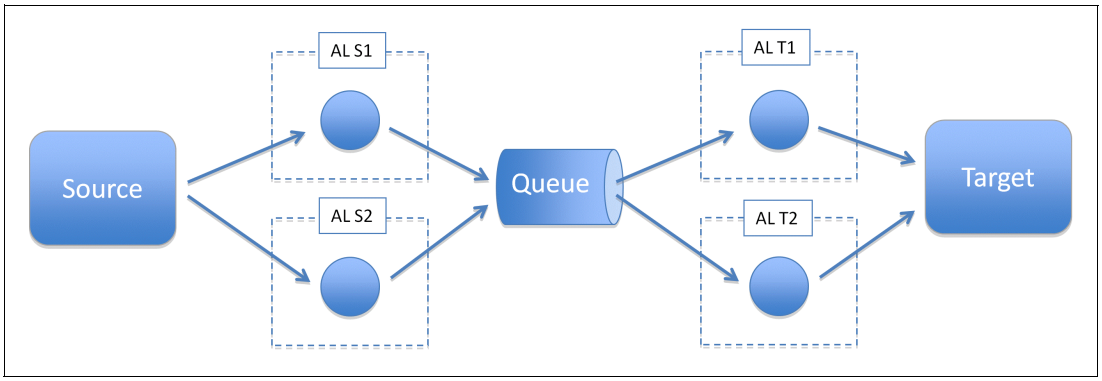


*Figure 3 Highly available solution using message queues*

To learn how to build such solutions, you can simulate this approach on a single server, even without the use of a highly available message queue. Tivoli Directory Integrator includes a memory queue that takes the place of the message queue in Figure 3. (The queue only works internally in each Tivoli Directory Integrator server, so it cannot be used for failover scenarios.) All four AssemblyLines S1, S2, T1, and T2 can run on the same server in the test environment.

For more information about the Tivoli Directory Integrator Memory Queue Connector see the Connectors[8] section in the *IBM Tivoli Directory Integrator Reference Guide Version 7.0*, SC23-6562.

"Starting AssemblyLines" on page 27 provides more information about the options for automatically starting multiple AssemblyLines.

# Monitor Tivoli Directory Integrator

Tivoli Directory Integrator executes either in a batch fashion, where the server shuts down when all work has been performed, or in daemon (or service) mode, where it stays alive even when no work is to be done. In the latter case, jobs can be started on Tivoli Directory Integrator in the following ways:

- Through the command-line tool, described in "Command-line interface" on page 29
- From the AMC, described in "Understanding the Administration and Monitoring Console" on page 18
- The JMX monitoring API
- Through custom code using the Tivoli Directory Integrator API

These mechanisms can also be used to monitor the status of Tivoli Directory Integrator, and change states that AssemblyLines can react to.

"Using events to signal activity, status, or trouble" on page 14 describes how Tivoli Directory Integrator solutions can emit and receive events using a number of protocols that can be intercepted by management and monitoring products.

[8] Available at Tivoli Directory Integrator 7.0 Information Center Web site (Memory Queue Connector): http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/referenceguide43.htm#memqueueconnect

## Understanding the Administration and Monitoring Console

The Tivoli Directory Integrator Administration and Monitoring Console[9] (AMC), shown in Figure 4, provides a Web interface to inspect and manage running Tivoli Directory Integrator servers. Other management systems can interface with Tivoli Directory Integrator also, but the AMC *understands* Tivoli Directory Integrator by design and needs minimal configuration to provide insight into the running Tivoli Directory Integrator solutions.

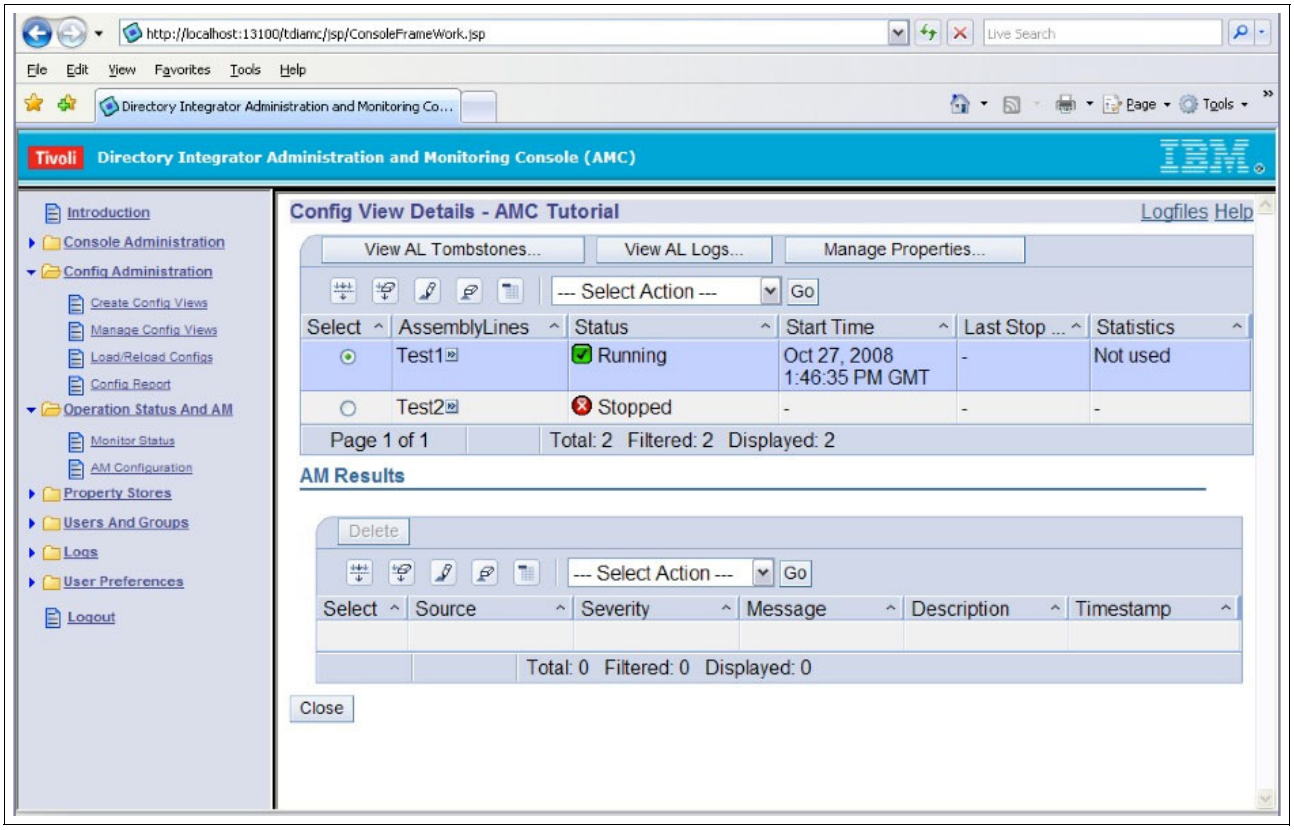


*Figure 4 The Administration and Monitoring Console Web interface*

The AMC can be deployed on WebSphere Application Server, but also on the embedded lightweight Web server platform for small deployments where WebSphere Application Server is not needed. The lightweight Web server platform is included in the Tivoli Directory Integrator distribution.

**Additional information available:** A helpful AMC deployment guide is available through the IBM Tivoli Directory Integrator Users Group at the following location:

`http://www.tdi-users.org/twiki/pub/Integrator/HowTo/TDI_AMC_Guide.pdf`

Multiple Tivoli Directory Integrator servers and solutions can be monitored simultaneously. Custom *health AssemblyLines* can be designated to populate custom data in the AMC console to provide integration-specific context to the AMC view.

[9] For more information about the AMC go to:
`http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide83.htm#ch_amc`

AMC contains a monitoring service called *Action Manager*[10], which allows construction of rules that are executed on certain conditions, enabling a first layer of availability to Tivoli Directory Integrator solutions. By design, Action Manager does not provide a failover service. Consider the high-level monitoring systems for integration scenarios with very high availability requirements.

AMC can monitor any number of Tivoli Directory Integrator solutions running on any number of servers. A single monitoring window can list the execution status of the selected solutions, and provide drill-down options for details.

Also, Tivoli Directory Integrator solutions can, with little effort, provide custom information to an AMC console so that the person monitoring the solutions can better understand what goes on in each integration job.

## Watching Tivoli Directory Integrator with monitoring systems

"Using events to signal activity, status, or trouble" on page 14 indicates that a number of mechanisms can be put into use when you consider how to integrate Tivoli Directory Integrator with a monitoring system. Tivoli Directory Integrator can emit events, which can be discovered and interpreted by external systems. Some of this happens by default (such as the logs, see "Locating and adding new log files" on page 21); other information must be emitted as part of the integration job itself, and must be inserted into the solution by those designing and developing it. Tivoli Directory Integrator can also handle requests from the outside. Several approaches use standardized interfaces, such as JMX or the Java API, requiring no consideration by the Tivoli Directory Integrator developer. Other approaches require actual development and configuration.

For example, AssemblyLines can use Server or Iterator mode to receive requests from other systems using SNMP, e-mail (POP or IMAP), HTTP/REST, Web Services, message queuing, and respond appropriately. For more information, see "Using events to signal activity, status, or trouble" on page 14.

### IBM Tivoli Monitoring using JMX

Tivoli Directory Integrator supports the JMX management protocol, which can be used to integrate Tivoli Directory Integrator with a number of products. In particular, it has been documented how to use this capability to integrate with IBM Tivoli Monitoring[11].

Review the following guide about the IBM Open Process Automation Library (OPAL). It describes how to integrate Tivoli Directory Integrator with IBM Tivoli Monitoring using JMX:

http://www.ibm.com/software/brandcatalog/portal/opal/details?catalog.label=1TW10TM78

Figure 5 on page 20 shows two IBM Tivoli Monitoring screen captures, which indicate seamless integration of Tivoli Directory Integrator.

[10] For more information about the Action Manager, go to: http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide88.htm#am

[11] For information about IBM Tivoli Monitoring, go to: http://www.ibm.com/software/tivoli/products/monitor/

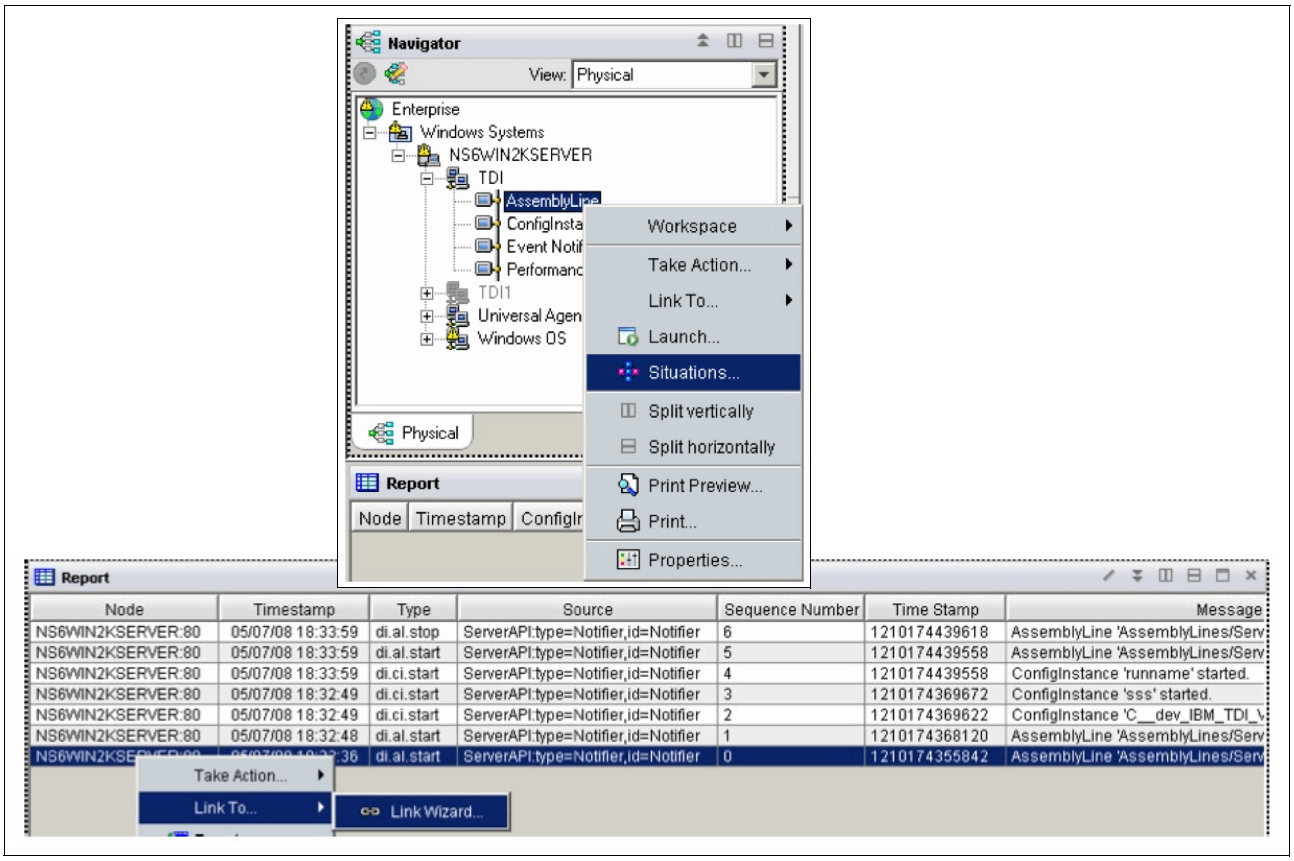


*Figure 5 IBM Tivoli Monitoring integration for Tivoli Directory Integrator*

# Planning for availability: tips and tricks

The power of Tivoli Directory Integrator is founded on the default automated behavior that reduces to a minimum the necessary tasks for a working solution. However, given the flexibility of Tivoli Directory Integrator and the few constraints it places on architecture and solution design, understanding how Tivoli Directory Integrator starts, runs, and terminates an AssemblyLine is important. To better understand this concept, see the “General Concepts - The AssemblyLine”[12] chapter in *IBM Tivoli Directory Integrator Users Guide Version 7.0*, SC23-6561.

To better follow our topics, become familiar with *AssemblyLine flow and Hooks*[13] as described in the appropriate section of *IBM Tivoli Directory Integrator Users Guide Version 7.0*, SC23-6561.

The Tivoli Directory Integrator pipeline is described in the following document. Take care to distinguish between the flow of an AssemblyLine and the flow of individual Connector modes.

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/TDI_7.0_FlowDiagrams.pdf

[12] Available at Tivoli Directory Integrator 7.0 Information Center Web site (The AssemblyLine): http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/theassemblyline.htm

[13] Available at Tivoli Directory Integrator 7.0 Information Center Web site (AssemblyLine flow and Hooks): http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/hooks.htm

## Locating and adding new log files

By default, the Tivoli Directory Integrator server logs everything to the following file:

```
<TDI solution directory>/logs/ibmdi.log
```

New loggers may be added, even at a more granular level, such as for each AssemblyLine. To add a new logger for an AssemblyLine, perform the following steps:

1. Click **Insert**, as shown in Figure 6.

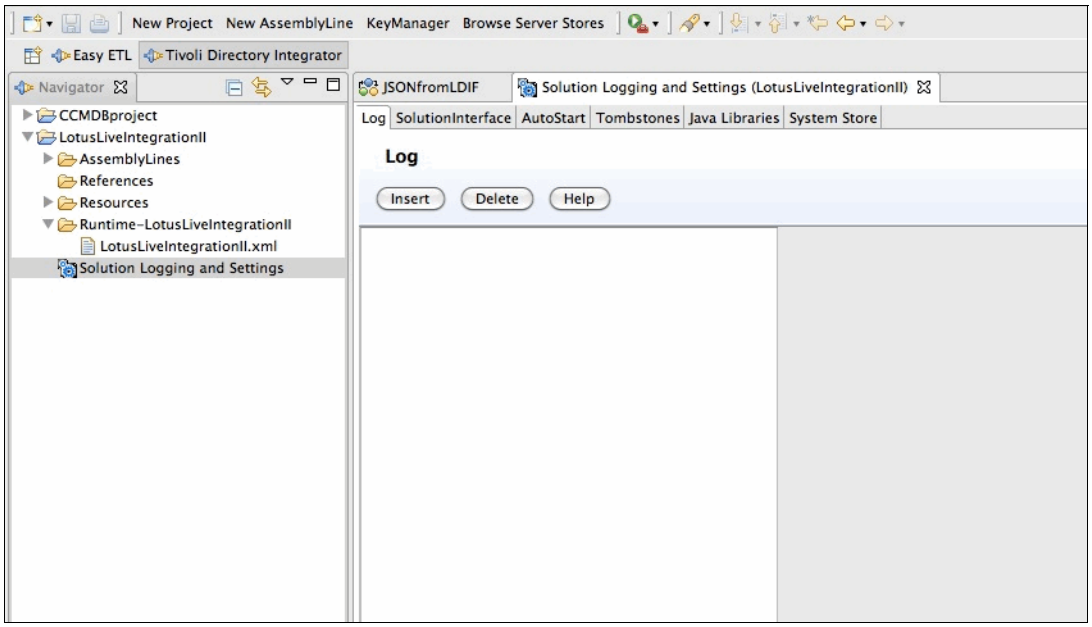


*Figure 6 Define a new logger*

2. With a new logger in place, select **Log Settings** to configure the AssemblyLine, as shown in Figure 7.

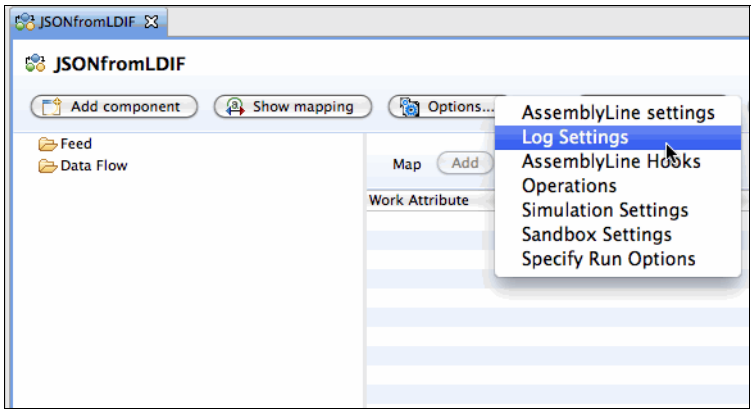


*Figure 7 AssemblyLine Log Settings*

3. Specify the additional information, as shown in Figure 8 and click **OK**.

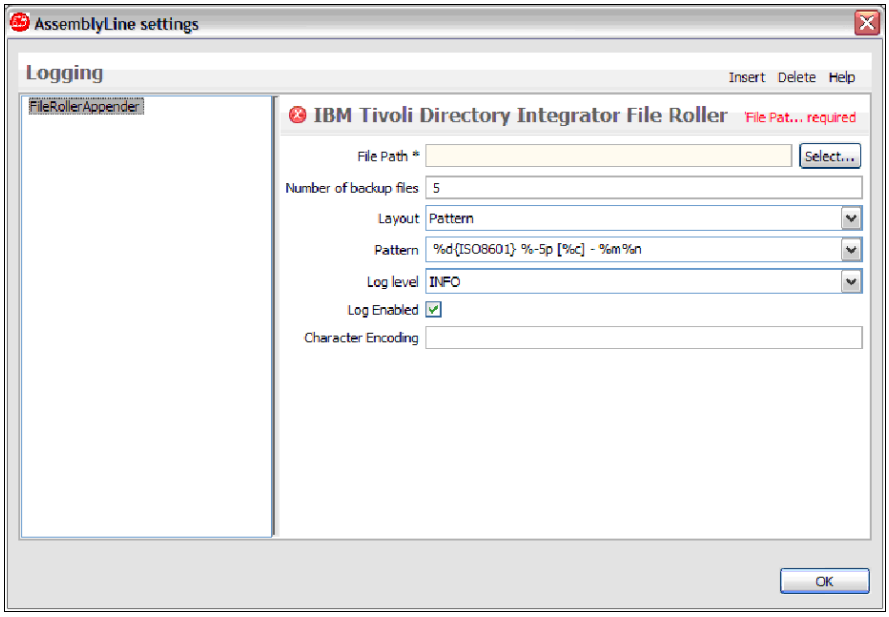


*Figure 8 Specify logging options*

For more information and Tivoli Directory Integrator documentation about logging, see the following resources:

- Logging and debugging:

  http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide74.htm#logging

- Log configuration:

  http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide76.htm#loggingconfig

- Log4j parameters

  http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide78.htm#logdef

## Minimum error handling in the Error Hook

This section describes the available error handling capabilities in more detail. However, a Tivoli Directory Integrator developer can do several steps to significantly improve the resiliency and maintainability of a Tivoli Directory Integrator solution.

Tivoli Directory Integrator provides *hooks* for customization. These points in the execution pipeline allow the developer to insert custom JavaScript that gets executed if the conditions for a hook are met. For example, *Error Hooks* can be called when certain errors occur. If no JavaScript has been placed into these hooks, then nothing will be executed, leading Tivoli Directory Integrator to look for a higher level Error Hook (for example at the AssemblyLine level), and ultimately to terminate the AssemblyLine if no Error Hook is found.

### Scripting Error Hooks

Tivoli Directory Integrator adds information to the log files when an AssemblyLine fails. That information indicates where the problem occurred and what kind of problem it was. The

information that Tivoli Directory Integrator logs might not be enough to understand the nature of the problem. By adding the following lines of script to the AssemblyLine Error Hook, detailed information will be available about the error object and *work*, which contains the attributes that the AssemblyLine was working on when the problem occurred:

```
task.dumpEntry(error);
task.dumpEntry(work);
```

The next level of logging can be done at the Connector level, which has its own set of hooks. In the previous paragraph, the AssemblyLine Error Hook has no insight into the component level that failed. Say, a lookup into a database failed. Tivoli Directory Integrator first checks for enabled Error Hooks for that Connector, finding none it would check for AssemblyLine level Error Hooks. By adding a `dumpEntry` call in the Connector Error Hook, Tivoli Directory Integrator also outputs the contents of the most recently used data object with that Connector:

```
Task.dumpEntry(conn);
```

After executing this Connector level hook, Tivoli Directory Integrator does not look for other hooks unless instructed to do so. By adding the following script to the Connector Error Hook, the AssemblyLine-level Error Hook is executed also, before the AssemblyLine is terminated:

```
throw error.message;
```

More information about error handling is in the document *Handling Exceptions/Errors with TDI* by Eddie Hartman. This document is on the IBM Tivoli Directory Integrator Users Group Web site:

http://www.tdi-users.org/twiki/bin/view/Integrator/HowTo

More information about hardening your integration solutions is in the *IBM Tivoli Directory Integrator Getting Started Guide Version 7.0*, GI11-8185:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/Hardeningyourintegrationsolutions.htm

## Details about scripting in Error Hooks

A little extra effort is valuable when logging information about the error. The *error* object is available when you are operating in an Error Hook. The error object is an Entry object similar to *work* or *conn*. The error object contains the following attributes:

| | |
|---|---|
| **connectorname** | The component that failed |
| **operation** | The operation/hook/attribute map that failed |
| **message** | The error message returned by the Tivoli Directory Integrator server |
| **exception** | The underlying exception that caused this issue, for example, thrown by the JDBC Driver or library API, or it might be a script error |
| **status** | The status, which holds the value *ok* until an error occurs |

At the very least, use `task.dumpEntry(error)` to record the details of the problem to your log output. If the error object is not available, such as in the catch-block of a try-catch, the following information is still available:

**component name** `ThisConnector.getName()`

**exception** The variable in the `catch(exc)`, in this example *exc*.

**situational information** Included context information about what this component is supposed to be doing. For example:

```
var url = "www.example.com";
try {
   var httpResponse = system.httpGet(url);
} catch ( exc ) {
   task.logmsg("ERROR", "Could not access "+url, exc);
}
```

## Configure automatic reconnection

Tivoli Directory Integrator provides an automatic connection retry capability that can be selected per Connector. It is intended for handling transient problems such as network or connectivity problems that can happen sporadically, but do not last for a long time. The Connection Error dialog box is shown in Figure 9.

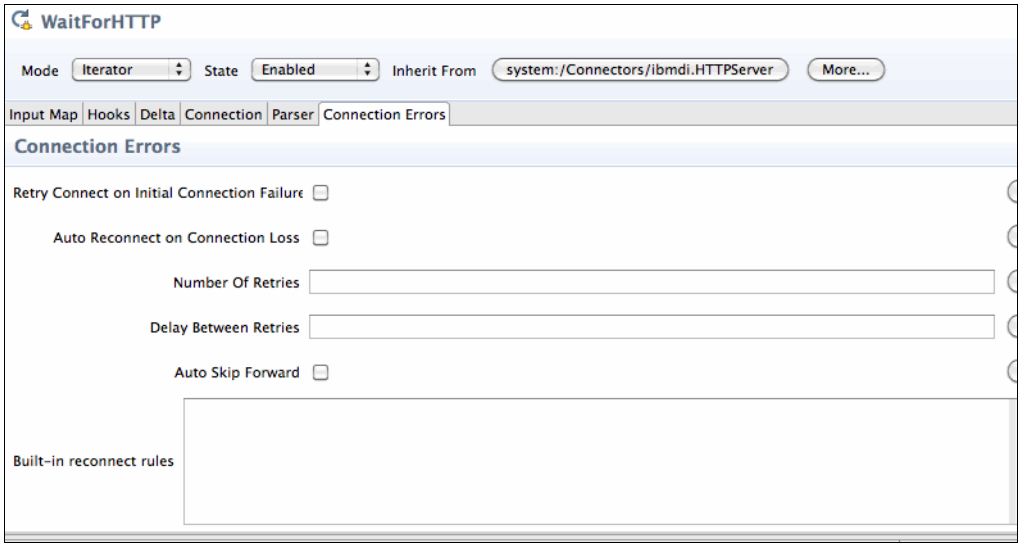


*Figure 9 Connection Error dialog box*

More information about connection errors is in the *IBM Tivoli Directory Integrator Users Guide Version 7.0*, SC23-6561. Connection Errors information is at the following location in the Information Center:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.IBMDI.doc_7.0/ce-reconnect.htm

The section “Connector Reconnect” in the Tivoli Directory Integrator pipeline document (*IBM Directory Integrator*, Hook Flow diagrams) describes the flow for these hooks and services; understand them before moving on to more advanced reconnection capabilities.

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/TDI_7.0_FlowDiagrams.pdf

## Failover to another data source

Tivoli Directory Integrator can be configured to change the Connector parameters and issue a re-initialization command. Although this configuration can be done in the *on connection failure* hook, the Tivoli Directory Integrator pipeline document (*IBM Directory Integrator*, Hook Flow diagrams) indicates that the configuration be executed before the automatic reconnection. Handling a reconnect of transient problems would then have to be handled in a script.

Rather, the failover should be done in the Connector's On Error Hook, which is called when the reconnect logic fails. By checking the error object implementing a failover solution, as shown in Example 1, is simple. This example changes the Connector parameters directly in the script.

*Example 1 Connection failure script*

```
// This is how we know that this is a connection failure problem
if error.operation.toString.startsWith("Automatic Reconnect") {
// Here would be optional code to check for failure of the backup server as well,
// meaning we've been here before, and optionally fail back to master
//
// set new connection parameters, assuming same credentials
thisConnector.getConnector().setParam("ldapUrl","ldap://myBackupServer.com")
thisConnector.reConnect ();
// at this stage one needs to consider the kind of operation that is being performed
// on the source or target. If it's an atomic add/delete/change, most of the below
// can be skipped.
// If this is an iterator, then it has to be considered if it's enough to
// start at the beginning again and redo all records once more, or skip forward by
// using a counter that has been kept track of.
// For all options, first the data set needs to be selected
thisConnector.getConnector().selectEntries();
// to optionally skip forward:
for (i=0;i<myCounter;i++) thisconnector.getConnector().getNextEntry()
//
// The call that redoes the operation against the target and should get us
// back in business again.
getNextEntry()
}
```

### Failing over in a more structured manner

As mentioned, the code shown in Example 1 changes the Connector parameters directly in the script. That approach makes debugging and testing unwieldy because any changes to the parameters must then be redone wherever they are used. Although using *properties* can reduce this inconvenience somewhat, a better approach is to use the Tivoli Directory Integrator *inheritance* capabilities (see "Connector library and inheritance" on page 33). With this approach, you may specify a failover Connector in the Resource library that can be managed in a single place, and use the generic code shown in Example 2 on page 26 to fail over to it.

For example, the two Connectors A and B point to the main and backup system. By dragging A into the AssemblyLine, the new Connector in the AssemblyLine *inherits* everything from A. Changes to A should be done in the Resource Library rather than in the AssemblyLine.

*Example 2 Generic failover script code*

```
ref = thisConnector.connector.getConfiguration().getInheritsFromRef();
task.logmsg("Failed: " + ref);
if(ref.equals("/Connectors/A")) {
   thisConnector.connector.getConfiguration().updateInheritsFrom("/Connectors/B");
}
thisConnector.reConnect();
```

How Error Hooks behave with the Automatic Reconnect feature can be confusing. In Tivoli Directory Integrator 7.0, the best practice for adding failover logic is to place it in the *on error* hook with the appropriate check for what kind of error it was, as described in the script code in Example 2. The reason for this is that an automatic reconnect will not be executed if code is placed in the *on connection error* hook.

### Skipping data that was previously read when failing over

When re-reading a data set, the Tivoli Directory Integrator *Delta engine* is a tool that is useful for skipping those records that have already been processed. This approach is also important when the selection criteria do not return the data sorted in the same way. An alternate approach to the Delta engine is to use the **Compute Changes** check box (in the Connector Update Mode panel) if the target system is configured in update mode. Tivoli Directory Integrator then checks the content of the target before writing any data, and it does not perform the update operation if the target already contains the values about to be written. Read more about these options in “Change Detection Connector” on page 32.

More information about the Delta mechanism in Tivoli Directory Integrator is in the “Deltas” Chapter of the *IBM Tivoli Directory Integrator Users Guide Version 7.0*, SC23-6561 at the following Web address:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/ch_delta.htm

## Automation with batch file

“Read from database and write a file report” on page 8 describes that custom exit code can be checked in a batch file, and that it can send alerts to administrators in case of trouble. The batch file examples in Table 2 on page 27 might initially seem excessive, but they set environment variables that other batch files need. If your Tivoli Directory Integrator script calls the `main.shutdownServer(4)` function in an Error Hook, the batch file to catch this can look like our examples.

*Table 2 Batch file automation for Windows and UNIX*

| Windows | UNIX |
|---|---|
| @echo off<br>setlocal<br>set TDI_INSTALL_DIR=C:\Program Files\IBM\TDI\V7.0<br>cd "%TDI_INSTALL_DIR%"<br>call ibmdisrv.bat<br>-c"C:\<path>\NameOfConfig.xml"<br>-r"NameOfAssemblyLine"<br>if "%ERRORLEVEL%"=="4" (<br>echo The TDI server was shut down by the AssemblyLine.<br>) else (<br>echo Server died unexpectedly.<br>)<br>endlocal | #! /bin/sh<br>TDI_INSTALL_DIR=/opt/IBM/TDI/V7.0<br>cd "$TDI_INSTALL_DIR"<br>./ibmdisrv [same parameters as for Windows]<br>if [ "$?" = "4" ] ; then<br>echo "TDI server shut down by the AssemblyLine."<br>else<br>echo " Server died unexpectedly."<br>fi |

Remember that sending e-mails and invoking command-line functions can be done from within Tivoli Directory Integrator also, as discussed in “Sending events” on page 15.

## Starting AssemblyLines

When you develop AssemblyLines, many ways are available to start and run them within your testing and debugging environment. That information is outside the scope of this document. The focus here is on how to start AssemblyLines from outside the development environment.

The Eclipse-based Tivoli Directory Integrator development environment automatically creates a *config file*. It contains all the AssemblyLines and components in a solution project, and is what a Tivoli Directory Integrator server expects to execute on.

### Server command line

The basic mechanism to start an AssemblyLine is to start the server with the `ibmdisrv.bat` batch file, using the following syntax:

```
ibmdisrv -c"C:\<path>\NameOfConfig.xml" -r"NameOfAL1" "NameOfAL2"
```

Tivoli Directory Integrator shuts down after the AssemblyLines are finished. Tivoli Directory Integrator can be instructed to keep running even when all AssemblyLines are completed. This technique can be useful when other systems start AssemblyLines at separate times, using the Tivoli Directory Integrator Connectors, the Java API, or the CLI tool as described in “Command-line interface” on page 29.

More information about the command-line options is in the “Command-line options” chapter of the *IBM Tivoli Directory Integrator Installation and Administrator Guide Version 7.0*, SC23-6560:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide72.htm#metamergeserver

### Autostart AssemblyLines and configs

A Tivoli Directory Integrator *config file* is the result of a Tivoli Directory Integrator development project, and is often a collection of AssemblyLines. It is a handy way of separating work because multiple config files can be loaded into a Tivoli Directory Integrator

server at the same time. Similarly, multiple AssemblyLines can run at the same time within one config instance.

Tivoli Directory Integrator looks in the autostart folder for any config files that should be loaded automatically. Also, use the AutoStart tab in the *Log & Settings* panel of your Tivoli Directory Integrator project (Figure 10) to specify the AssemblyLines that are to be automatically started each time Tivoli Directory Integrator starts.

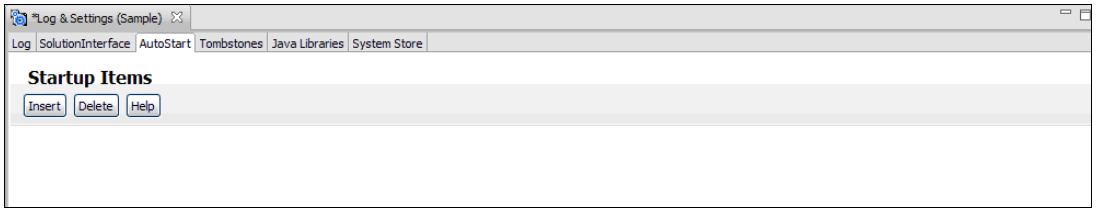


*Figure 10 Configuring Startup Items*

## Script-driven AssemblyLine and checking the status

Tivoli Directory Integrator provides components for launching AssemblyLines from within an AssemblyLine. For example, in AssemblyLine A you can launch AssemblyLine B. You may choose to wait for B's completion before continuing the execution of A, or you may *fire and forget* B and seamlessly continue A. However, sometimes a higher level of control is needed. Example 3 shows how an AssemblyLine named `AssemblyLineName` is executed with a minimum amount of checking for success or failure.

*Example 3 Script driven AssemblyLine start*

```
var al = ci.startAssemblyLine("AssemblyLineName", true);
var stats = al.getStatistics();
var exception = stats.getError();
if (exception == null) {
   task.logmsg("completed successfully. Records added " + stats.add);
} else if (exception instanceof java.io.IOException) {
   task.logmsg("connection failed: "+exception);
} else {
   task.logmsg("failed: "+exception);
}
```

A statistics object of the class `com.ibm.di.server.TaskStatistics` provides information that helps you inspect the results of the AssemblyLine that you started with the `ci.startAssemblyLine` call. It contains valuable information such as how many Entry objects an Iterator in the AssemblyLine have read, how many Entry objects the Connectors in AddOnly mode have written, and so on.

To verify whether an AssemblyLine is running, use the script in Example 4.

*Example 4 Is AssemblyLine running?*

```
var isRunning = false;
for (var al in session.getAssemblyLines()) {
   if ("AssemblyLines/myAssemblyLine".equals(al.getName())) {
      isRunning = true;
      break;
   }
}
```

### AssemblyLine Connector

Starting AssemblyLines on the same or a separate Tivoli Directory Integrator server from within an AssemblyLine by using the *AssemblyLine Connector* is simple. In certain availability and failover scenarios, Tivoli Directory Integrator can stay in control and manage other Tivoli Directory Integrator servers that do the actual work. Although the AssemblyLine can be started and left alone, waiting for their termination and use exit information to make further decisions is also possible. Finally, the target AssemblyLine can be used as a function, repetitively calling it with new parameters. Parameters may be passed into the called AssemblyLine that can be used to set up local Connectors, or in other ways to dynamically drive the behavior of the target AssemblyLine. More information about the AssemblyLine Connector is in the “Connectors” chapter in the *IBM Tivoli Directory Integrator Reference Guide Version 7.0*, SC23-6562:

`http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/referenceguide12.htm#aliconnect`

### Command-line interface

The *command-line interface* or *CLI* (`ibmdisrv.bat`) is used to start the Tivoli Directory Integrator server. However, the Tivoli Directory Integrator server might already be running, or it could have been started as a service under Windows or UNIX, possibly in *daemon mode*. You may use another command-line tool, named `tdisrvctl`, which uses the Tivoli Directory Integrator API to communicate with any running Tivoli Directory Integrator server. This tool is handy when you want to control Tivoli Directory Integrator on another server.

More information about the CLI is in the “Command-line options” chapter in the *IBM Tivoli Directory Integrator Installation and Administrator Guide Version 7.0*, SC23-6560 at the following location:

`http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide73.htm#cli`

In addition to running and stopping an AssemblyLine, you can use `tdisrvctl` to set properties within Tivoli Directory Integrator that can change the behavior of Connectors. A conditional IF component in the AssemblyLine can check for a property value and execute a script such as the one shown in “Failover to another data source” on page 25. A remote system can then fail-over the Connectors in an AssemblyLine to other systems by using a single **`tdisrvctl`** command.

### Web-based administration and monitoring

The Administration and Monitoring Console (AMC) is described in “Understanding the Administration and Monitoring Console” on page 18. It provides a Web interface in which AssemblyLines can be monitored, started, and stopped. The Action Manager can execute rules that monitor behavior and provide a number of features to control and run AssemblyLines.

More information about the AMC and Action Manager is in the “Administration and Monitoring” chapter in the *IBM Tivoli Directory Integrator Installation and Administrator Guide Version 7.0*, SC23-6560:

`http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide90.htm#amcui`

## Sending events from Tivoli Directory Integrator

In addition to performing the actual integration, all of the connectivity options in Tivoli Directory Integrator can be used for alerting, logging, or any other purpose to reduce the cost

of managing the solution. Even if the solution works well after it is deployed, all enterprise architectures change over time, which affect the Tivoli Directory Integrator solution. Certain organizations use monitoring products to continuously monitor selected parts of their IT infrastructure. Although these tools are capable of warning IT staff and automating support actions, many organizations have not deployed tools with this capability.

Tivoli Directory Integrator can provide various built-in capabilities for monitoring and logging. See “Monitor Tivoli Directory Integrator” on page 17 and “Locating and adding new log files” on page 21. Log files can be inspected by other systems to determine the source of trouble and initiate appropriate actions on the issues that can be recognized.

Tivoli Directory Integrator provides Connectors for SNMP, e-mail (SMTP), JMX, command-line (to invoke command-line based services), HTTP/REST, Web services, and many more that can be used to inform other IT systems in the organization about work in progress and status. Similar Tivoli Directory Integrator components can be used to receive incoming requests for information and provide a status about running AssemblyLines.

For more details about this subject, see “Sending events” on page 15.

### Changing Tivoli Directory Integrator behavior from the outside

When the Tivoli Directory Integrator server has loaded a config file and started the AssemblyLines there are still many ways to control and change its behavior.

The Tivoli Directory Integrator Web-based administration and monitoring interfaces are described in “Understanding the Administration and Monitoring Console” on page 18. Also, understand the run-time command-line tools in “Command-line interface” on page 29. A Java API is also available, to interface tightly with Tivoli Directory Integrator.

Both properties and events can be set in (or sent to) a Tivoli Directory Integrator server to change the behavior of AssemblyLines.

## Change detection

Although detecting changes might seem like an easy process, it is a surprisingly challenging subject. Choosing the correct approach can greatly affect performance and availability, and also the development, testing, and maintenance effort.

Tivoli Directory Integrator provides a set of tools and services that can be applied to practically any scenario. In this section, we describe these capabilities and provide examples to illustrate their relevance. Each capability is not necessarily directly relevant in terms of availability, but understanding them can provide a clearer picture of how Tivoli Directory Integrator can be used for resilient and efficient integration solutions.

### What are the changes

The optimal situation is when the source system can inform of, or be queried for, changes that have happened within an interval of time. Then it is a simple matter of reading those changes and driving them to the target in the appropriate fashion. However, sometimes the old values are needed to determine the correct action. Certain systems provide this; others provide only a pointer to the new and updated record. Sometimes only changes in specific attributes are relevant, yet many systems cannot provide that information. Do not forget the data that has been deleted in the source system. Certain systems cannot provide any change information

at all. This is even the case with systems that technically can, but the business does not. For example, SAP can inform of changes perfectly well, but the administrators might choose to provide only a database export that is merely a snapshot of the current data, which is not much to go on.

With this information in mind, we can now look at the built-in capabilities that Tivoli Directory Integrator provides to manage change. Which pieces are put to work depends entirely on the characteristics of the problem.

**Additional material:** For an in-depth study on this subject, read the tutorial that is available at the following Web address. It is written for Tivoli Directory Integrator 6.1.x, but everything is true for Tivoli Directory Integrator 7 also.

http://www.tdi-users.org/twiki/pub/Integrator/HowTo/HowTo_SyncData_6.1.1070523.pdf

## Compute changes in target

Writing to a target system consumes more time and resources than simply reading from it. Tivoli Directory Integrator can optionally attempt to read the target data before writing to it. This approach allows Tivoli Directory Integrator, when configured in update mode, to automatically determine if an *add* or an *update* operation is needed. Furthermore, Tivoli Directory Integrator can check the existing data on the target system against what is ready to be written to it. If the values are equal, Tivoli Directory Integrator does not perform the write operation at all. This option is also suitable in situations where updating records must be kept to a minimum because of downstream workflows that get initiated when data changes on the target.

From an availability perspective, this approach (of determining the state of the target before actually updating) adds no complexity to a Tivoli Directory Integrator solution. In an environment with a limited data set, the best approach can be to read through the entire source every so often and perform an update operation against the target by using the *compute changes* option. This approach can even be done on two identical Tivoli Directory Integrator servers, providing a complete failover environment. The cost can be measured in extra read operations, performed on the source and target, but that might be a fully acceptable cost given the otherwise simple and cheap solution.

## Delta engine

Tivoli Directory Integrator provides an optional service that uses a back-end database (of your choice) to store snapshots of data that has passed through a Connector in Iterator mode. The benefit of this service is that it works automatically with the Iterator. Every time the Iterator connects to the data source (such as a file), it automatically skips those records that have not changed since the last time it was read. Any new or changed records are passed to the AssemblyLine and updated in the Delta engine. Records are also tagged with Tivoli Directory Integrator meta information to indicate a changed or a new record to the AssemblyLine. For more information about this topic, see “Delta Entry” on page 32. Finally, when everything in the source has been read, the Delta engine can determine what records did not come through in this iteration cycle, but did the last time. These records are then passed to the AssemblyLine as though they had been read from the source, but tagged as *deleted*, so that the AssemblyLine code can perform the appropriate action.

Despite this somewhat intricate description of the Delta engine, it is very simple to configure, and can help you quickly determine changes on data.

### Delta Entry

This section is important if change propagation is in your plans. The *delta Entry object* (DE) lets Tivoli Directory Integrator automatically handle otherwise complex logic. The DE is an important element in all of the Tivoli Directory Integrator change services. Although using a DE to create integration solutions with Tivoli Directory Integrator is not necessary, understanding and using DE can significantly improve the quality of the solutions and increase the performance.

Delta Entry describes a Tivoli Directory Integrator Entry data structure that has been tagged with additional meta information that makes it suitable for more advanced and automatic consumption by other components. Change components add tags to the data so that other components can inspect it and make better decisions.

### Change Detection Connector

The *Change Detection Connector* (for sources like Microsoft Active Directory and Lotus Domino) differs from the Changelog Connector in that it returns the entire record from the source system after modification. There is no information about what has been modified in the record, or what the old values were. This might not be a problem for the given business scenario, but if that information is needed, use the Delta engine in conjunction with the Connector to add this logic to it at a performance cost.

### Changelog Connector

The *Changelog Connector* (for sources such as IBM Tivoli Directory Server, Sun Directory, and databases with triggers) returns full delta Entry object with all the information necessary to be used with Delta mode.

### LDIF parser

LDAP directories use the *LDIF format* to report changes to other directories. The format includes a tag standard that allows other directories to import the LDIF file from another server and automatically determine new, changed, and deleted records, including change information down to the attribute value level. Tivoli Directory Integrator can read and write this format directly based on delta Entry object, dramatically simplifying the creation and consumption of LDIF formatted data.

### Delta mode

The *Delta mode* is supported by the LDAP and the RDBMS Connectors. It requires a delta Entry created by the Delta engine, a true changelog Connector or a parser that delivers such records (LDIF). A non-true changelog Connector must use the Delta engine to retrieve the records tagged appropriately, because we need the old value to compare.

## Proper Tivoli Directory Integrator design and scripting

In this section, we cover general good practices when developing Tivoli Directory Integrator solutions. We also describe Tivoli Directory Integrator capabilities that can be helpful when developing for resilience. This section is more focused on development than “Planning for availability: tips and tricks” on page 20, although both sections can be helpful to anyone who

plans to develop a solution with availability requirements. Although most topics described in this section are for the experienced developer, many Tivoli Directory Integrator integration developers who have little or no traditional development background might also find the information in this section helpful. The relevance to resilience and availability is that integration solutions can become more robust, and therefore, can reduce the chances of unexpected failures when deployed.

A very good source for learning is the Tivoli Directory Integrator online help system. It contains all of the manuals and it provides full-text search capabilities. Several steps can configure the system to search only the Tivoli Directory Integrator 7.0 documentation. Click **Search scope**, and then configure a scope; the system *remembers* that for future searches (Figure 11).

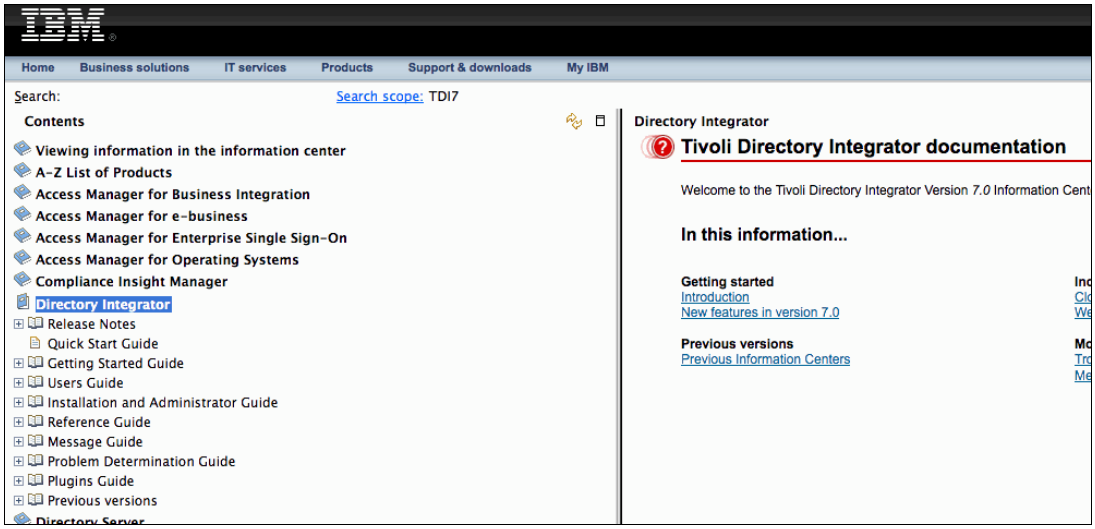


*Figure 11 The Tivoli Directory Integrator online documentation*

The Tivoli Directory Integrator online documentation can be found at:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.IBMDI.doc_7.0/welcome.htm

## Adding automatic reconnection to Connectors

This topic is described in more detail in “Configure automatic reconnection” on page 24.

## Connector library and inheritance

With a simple drag-and-drop operation, a configured Connector can be shared (inherited) across multiple AssemblyLines. That includes the configuration attributes, attribute mapping, and any custom JavaScript in the hooks. In this way, modifications can be done in a single place. This topic is covered in more detail in the “Using Lookup Mode” section in the *IBM Tivoli Directory Integrator Getting Started Guide Version 7.0*, GI11-8185:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/Usinglookupmode.htm

Inheritance is not limited to Connectors, although that is what we discuss here. By placing a Connector into the Connector library (using the drag and drop method), it can be pulled out again for use in other AssemblyLines. Inheritance can be *broken* (which means that a specific element is not inherited) for those parts of the Connector where the common logic is not

appropriate. A Connector inherits from the system templates, and the *Inherit From* field, shown in Figure 12, can change after inheriting from a Connector in the Library. Click the **More** button to add a new line with more information; click **Inheritance** to provide more detail, as shown in the Configure inheritance dialog box.

**More or Less:** After you click the **More** button, it becomes a **Less** button.

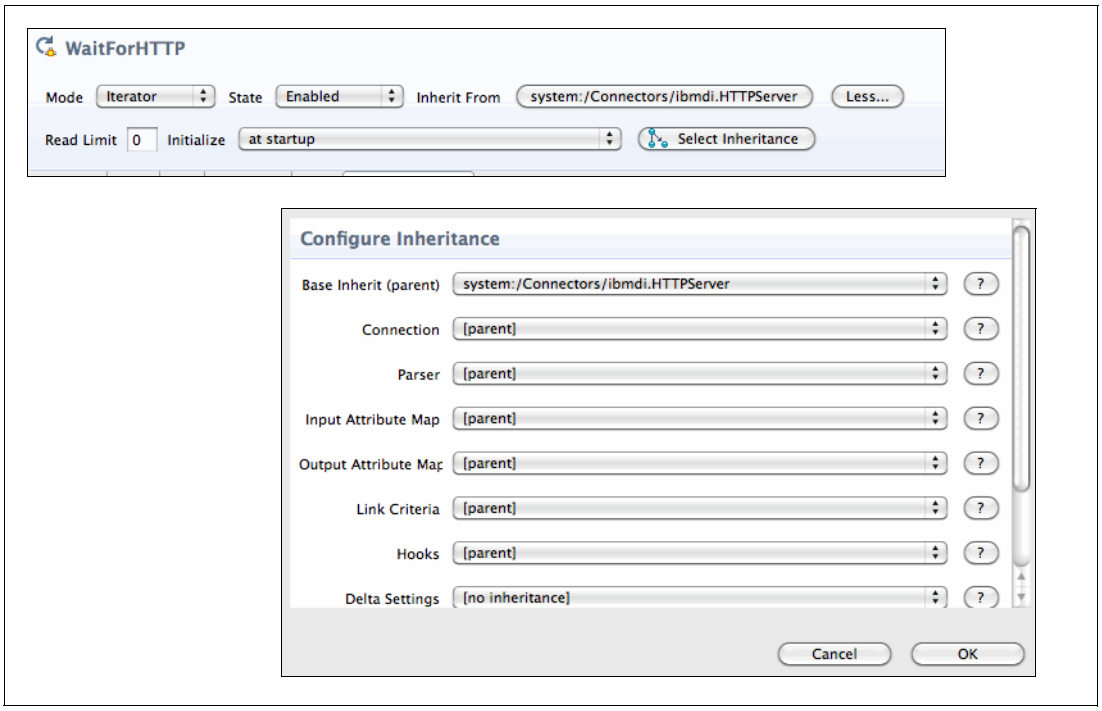


*Figure 12 Configure inheritance*

## Understanding the Tivoli Directory Integrator pipeline logic

The Tivoli Directory Integrator is a helpful tool to understand. It executes AssemblyLines according to a strict set of rules, as it does for Connectors and all its other components. Understanding this flow of events facilitates planning and developing solutions, and can help you with the debugging solutions in the Tivoli Directory Integrator debugger.

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/TDI_7.0_FlowDiagrams.pdf

## Adding simple debug AssemblyLines

Inherited Connectors can help you with the debugging process. Too often, integration solutions are developed as one large AssemblyLine rather than splitting it up for simpler testing. The simplest approach is to add a new AssemblyLine for each Connector in your solution and add the basic lines of script described in “Minimum error handling in the Error Hook” on page 22.

These AssemblyLines can then be executed from the command line, as shown in the following example:

```
ibmdisrv -c"C:\demos\NameOfConfig.xml" -r"NameOfAL"
```

More information about the command-line options is in the *IBM Tivoli Directory Integrator Installation and Administrator Guide Version 7.0*, SC23-6560:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/adminguide72.htm#metamergeserver

With a little more work, the test AssemblyLine can iterate on the first few records and display the output to the screen. This approach can help you immediately verify that the Connector is working and reading what it should. To implement this approach, add the following single line of script to the *after getnext* hook:

```
Connector: task.dumpEntry(conn)
```

There is a simple trick for this implementation in the AssemblyLine settings. In the Max number of reads (Iterator) field, shown in Figure 13, specify the maximum number of read iterations to perform before stopping. Set this value to a low number; the test AssemblyLine displays only those records and then stop.

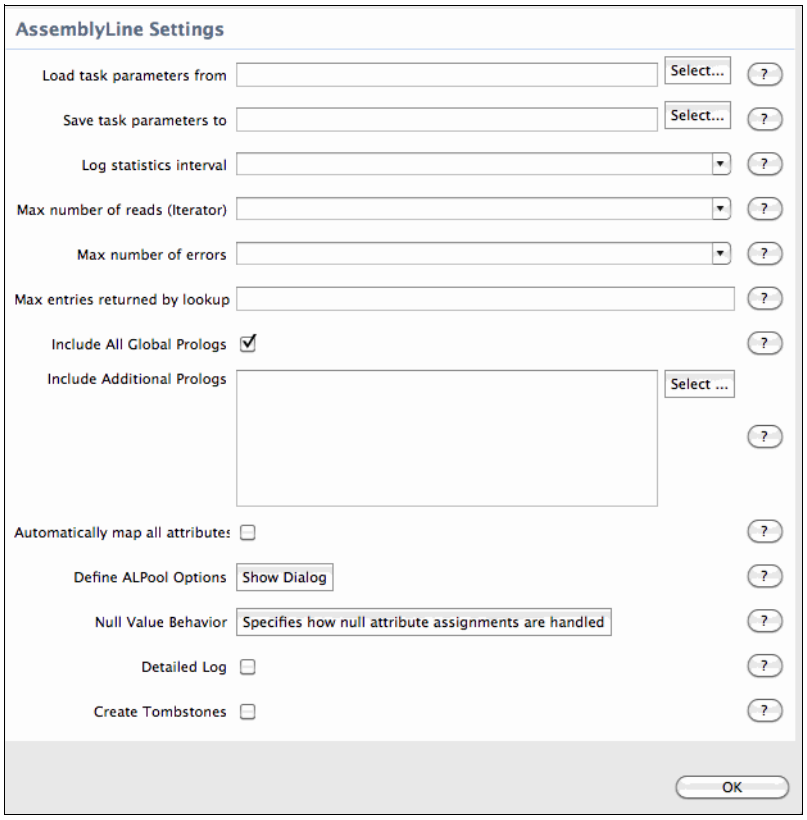


*Figure 13 Configuring the debug AssemblyLine*

## Missing data in source and null-value behavior

Unexpected data is a major reason for failure in many applications. The source system from which Tivoli Directory Integrator reads might have missing attributes, or attributes with data that differs from what is expected. Use Tivoli Directory Integrator to specify what to do when certain attributes do not exist, or are empty (Figure 14 on page 36). Although this approach is not enough to safeguard the integration solution, it does remove a significant amount of error-checking that otherwise would have been necessary.

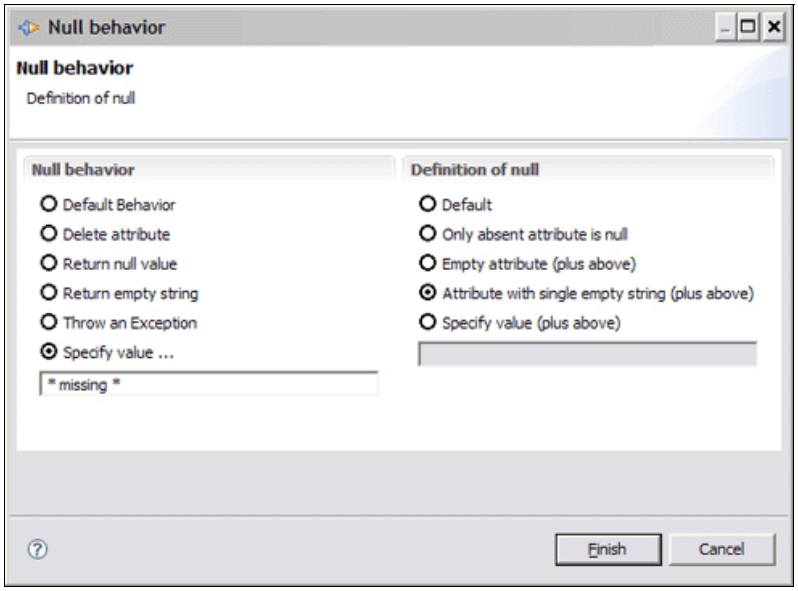


*Figure 14 Null behavior specification*

Four more information about null behavior, see *IBM Tivoli Directory Integrator Getting Started Guide Version 7.0*, GI11-8185:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/Nullbehavior.htm

## Handling unexpected data

Many problems occur because developers have made assumptions that might not be accurate. When thinking about availability and reliability, a healthy assumption is *if anything can go wrong, it will*.

Assuming that a certain field in a database will always contain digits can result in failure if this field must be converted to an integer, for example. Similarly, a slight mistake in a text field might result in difficulty of converting it to a date format. Division by zero is another classic example, and there are many more examples.

Java (and JavaScript) provides mechanisms for error handling. Although there are too many of them to cover in this document, several examples are helpful to understand the concepts.

*Exceptions* are *thrown* when processing goes wrong. These exceptions can be caught early by custom code, or higher level services. Unless caught early, the information available at the higher level might not be detailed enough to be of value. Example 5 illustrates catching errors early.

*Example 5 Using exceptions*

```
try {
   p = system.getTDIProperty("PropertyName"); // if this fails, it will drop into catch
   if (p == null) throw exception "nullValue"; // throws it into the catch
} catch (e) {
// The try failed, figure out why
   if ( e.getMessage() == "nullValue") {
      // I know it was empty, and not some other error
      task.logmsg("Error: Property 'PropertyName' is not set")
   // could set a default value, and continue developing
```

```
   } else {
      task.logmsg("Error looking for property 'PropertyName': " e.getMessage() );
      // log error message
      // may continue processing, or set a custom exit code and terminate
      // another option is to rethrow
      throw e.getMessage() (AssemblyLine will catch it if nothing else does before that
   }
// things look ok if we get this far
```

The code in Example 5 on page 36 is rather convoluted; it illustrates how an exception might be thrown. A cleaner implementation is shown in Example 6.

*Example 6 Implementation of throwing an exception*

```
try {
p = system.getTDIProperty("PropertyName");
} catch (e) {
task.logmsg("Error when looking for property 'PropertyName': " e.getMessage() );
// could continue processing, or set a custom exit code and terminate
// another option is to rethrow
throw e.getMessage(); // AssemblyLine will catch it if nothing else does before
that
}

if (p == null) {
task.logmsg("Error: property 'PropertyName' is not set");
// could call another AssemblyLine or shut down
// if you continue, make sure you set prop to something that will not lead to
// failure in the code that follows
}
```

All Java methods are available. Several examples are as follows:

```
work.OS.getValue().startsWith("Windows");
conn.email.getValue().contains("ibm.com");
work.cn.getValue().toUpperCase();
```

Typing `getValue().` in the Tivoli Directory Integrator script editor brings up available alternatives. Another option is to use the `java.io.*` library of functions that provide helpful functions that can be used to format and work with strings. Click the **Javascript Help** button in Tivoli Directory Integrator to learn more.

### Learning about the debugger

The Tivoli Directory Integrator debugger is another useful tool. The IBM Education Assistant offers online video education courses that provide a good overview of the capabilities of the debugger. The debugger can significantly reduce the time you spend searching for bugs and watching Tivoli Directory Integrator at run-time:

http://publib.boulder.ibm.com/infocenter/ieduasst/tivv1r0/index.jsp?topic=/com.ibm.iea.tdi/tdi/7.0/configuration_editor.html

## Reusing a Connector is not the same as inheritance

Unless directed otherwise, every Connector in an AssemblyLine fires up its own session with the target system. When developing an integration solution that is required to perform multiple operations against a target throughout an AssemblyLine, reusing a Connector means that

only a single instance of the Connector connects to the target. After a Connector has been defined in an AssemblyLine, it becomes available for reuse in the Connector listing. This technique can reduce the startup cost of each AssemblyLine as well as reduce the resource requirements on the target system.

## Logging

Logging is a general term, but to many people it refers to a system's *paper trail* so that unraveling what has happened earlier is possible.

The more information that is available after the problem occurs, the easier it is to uncover the sources of problems. To add output to the default log, you can simply add a script:

```
task.logms("Encountered a problem with record number " + work.RecordNumber)
```

Tivoli Directory Integrator logs a certain amount of information automatically. That information is generic Tivoli Directory Integrator-related information and several basic statistics after each AssemblyLine has executed. Tivoli Directory Integrator also logs errors that are reported from the Connectors. For more information about these logs, see "Locating and adding new log files" on page 21. The default logger can be changed to other available loggers; you may also write your own custom loggers.

The loggers can be used to alert monitoring systems that scan the output of certain standard logging types, such as UNIX syslogd and Windows Events, both of which are supported by Tivoli Directory Integrator.

Finally, logging can be used for business audit purposes, where you may review the Tivoli Directory Integrator logs to verify what records where written at what time. This step requires that the Tivoli Directory Integrator solution adds the appropriate output to the log files as described in "Locating and adding new log files" on page 21.

## Knowing what is in property files

The concept of *properties* is a powerful mechanism that allows values to be set in Tivoli Directory Integrator, based on content in external files or other systems from which Tivoli Directory Integrator can read properties. For example, credentials for a given Connector can be stored in a password-protected property file where it is easy to maintain, rather than having to go back into the Tivoli Directory Integrator solution to change it.

People make mistakes, and sometimes developers assume things they should not. Do not assume that users enter values into property files according to your expectations. Another source of problems is when a Tivoli Directory Integrator solution has been moved to another location without moving or updating the required property files too.

Several steps can reduce potential problems. In the Connector configuration panel, any parameter can be computed. In Figure 15 on page 39, the question mark (?) to the right of each parameter, when clicked, opens the Expression Editor. By selecting **Use Property**, you may choose a single property name. A better way is select **Advanced** and enter a script, as the one shown in Figure 15 on page 39. Also see, Example 7 on page 39.

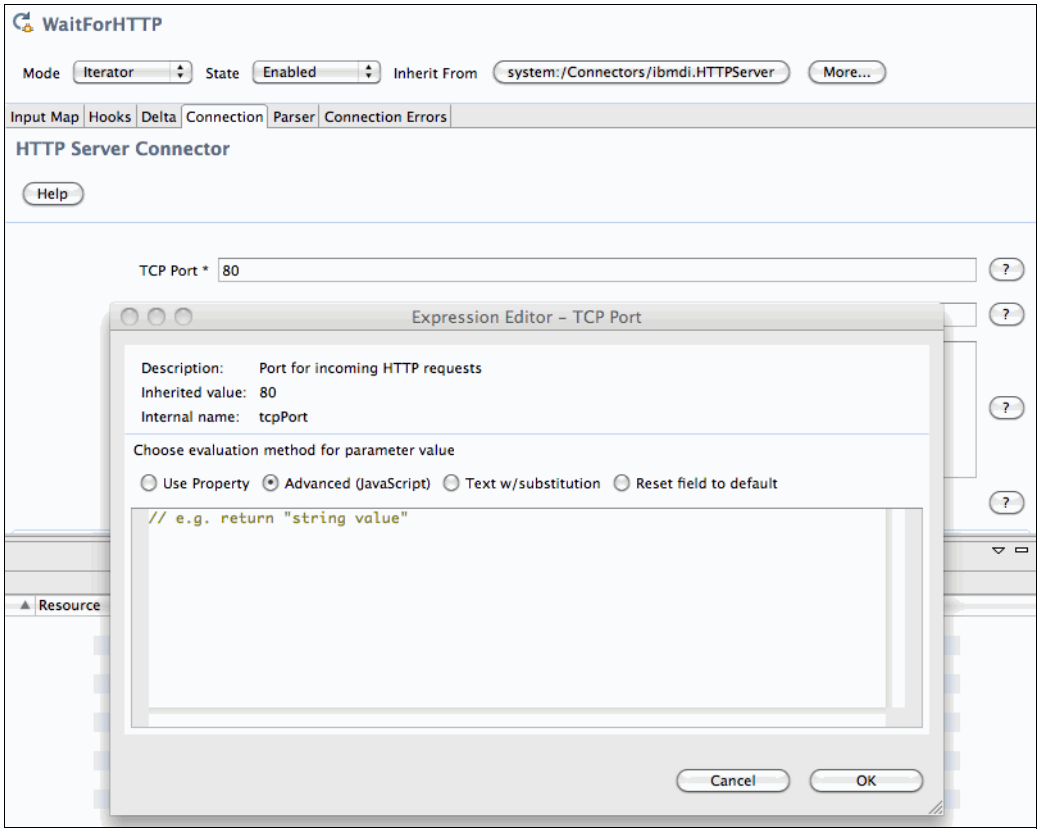


*Figure 15 The Expression Editor*

*Example 7 Advanced expression*

```
myURL = system.getTDIProperty("myURLprop");
if (myURL == null | myURL == "") {
task.logmsg ("Error message: Blank URL in the property 'myURLprop' ");
main.shutdownServer(18); // choose something that you catch in a batchfile
} else
return myURL
```

“Missing data in source and null-value behavior” on page 35 has more examples of error handling code. Missing properties is a common source of problems. If the solution depends on properties in an external property file, it can fail if it cannot find the file. Therefore, be sure to check for this in your code, as shown in Example 8.

*Example 8 Checking for properties*

```
prop = system.getTDIProperty("PropertyName");
if (prop == null) {
task.logmsg("Error: property PropertyName is not set");
// could call another AssemblyLine or shut down
// if you continue, make sure you set prop to something that will not lead to
// failure in the code that follows
}
```

# Summary

In this document, we have provided an overview of developing robust solutions with Tivoli Directory Integrator. The issues are no different from working with other tools, but there is a difference in how much tools allow the developer to cater to a wide range of technical scenarios. The more parts that exist in an infrastructure, the more susceptible the infrastructure is to problems. What was stable yesterday might not be stable tomorrow. Assuming otherwise, might cause problems.

# Other resources for more information

For additional information, see the following resources:

- Tivoli Directory Integrator product documentation:
  - *IBM Tivoli Directory Integrator Getting Started Guide Version 7.0*, GI11-8185
  - *IBM Tivoli Directory Integrator Users Guide Version 7.0*, SC23-6561
  - *IBM Tivoli Directory Integrator Reference Guide Version 7.0*, SC23-6562
  - *IBM Tivoli Directory Integrator Installation and Administrator Guide Version 7.0*, SC23-6560
  - *IBM Tivoli Directory Integrator Problem Determination Guide Version 7.0*, GI11-8186
- Tivoli Directory Integrator Information Center:

  http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.0/welcome.htm
- Community support:

  http://groups.google.com/group/ibm.software.network.directory-integrator/topics?gvc=2
- The IBM Tivoli Directory Integrator Users Group:

  http://www.tdi-users.org/twiki/bin/view/Integrator/WebHome
- Product page, support and fixpacks:

  http://www.ibm.com/software/tivoli/products/directory-integrator/
- General information about Tivoli Directory Integrator 7.0, which contains lots of links and video tutorials:

  http://sites.google.com/site/tdi7islive/
- The IBM Open Process Automation Library:

  http://www.ibm.com/software/brandcatalog/portal/opal

# The team who wrote this paper

This paper was produced by a team of specialists from around the world.

**Axel Buecker** is a Certified Consulting Software IT Specialist at the ITSO, Austin Center. He writes extensively and teaches IBM classes worldwide on areas of software security architecture and network computing technologies. He holds a degree in Computer Science from the University of Bremen, Germany. He has 23 years of experience in a variety of areas related to workstation and systems management, network computing, and e-business solutions. Before joining the ITSO in March 2000, Axel worked for IBM in Germany as a Senior IT Specialist in Software Security Architecture.

**Johan Varno** is the Lead Architect for IBM Tivoli Directory Integrator at the IBM Oslo Development Lab in Norway. He holds a degree in Computer Science from the University in Oslo and an MBA from the Norwegian School of Management. He has 28 years of experience in a variety of areas relating to network technologies, software development, and business development. Prior to working in IBM, Johan was cofounder and CTO of Metamerge®.

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

Diane Sherman
International Technical Support Organization, Austin Center

Eddie Hartman, Bjorn Stadheim
IBM

# Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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

**ibm.com**/redbooks/residencies.html

## Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks® publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

# 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.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the 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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

This document REDP-4672-00 was created or updated on June 1, 2010.

Send us your comments in one of the following ways:
- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD  Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.


# Trademarks