TIBCO iWay® Service Manager

Troubleshooting and Debugging Best Practices

Version 8.0 and Higher
March 2021
DN3502297.0321

Contents

Chapter 1

Defining Troubleshooting and Debugging Strategies

This section provides an introduction to iWay Service Manager (iSM) troubleshooting and debugging facilities, and outlines key strategies that can be used.

In this chapter:

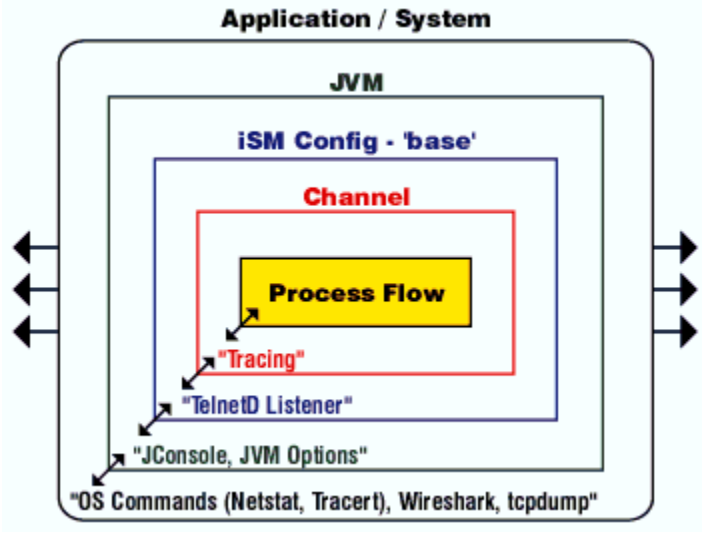
Troubleshooting and Debugging Overview

iWay Service Manager (iSM) provides a collection of tools and facilities that are designed for troubleshooting and debugging purposes. One of the key challenges in any debugging scenario is to identify the level (or area) where the error is occurring. As a starting point, asking the following questions can help during the early stages of an investigation:

- Is the application generating an error at the server level?
- Is the application generating an error at the channel? Should I be debugging within a channel?
- Is there an issue with the process flow(s) being used by the application? Should I be debugging within a specific process flow?
- Should I be debugging at a thread level?
- Is the error occurring during application design time or runtime?

In actual scenarios, issues may not occur at a specific level, but between levels. For example, as a message or transaction passes between the process flow and channel levels.

Based on the answers to these questions, the appropriate troubleshooting or debugging tool can be identified. The following diagram illustrates this logic and the various levels that a typical application may traverse.



The primary goals of this best practices guide is to:

1. Identify the troubleshooting and debugging facilities that are provided by iSM.
2. Identify iWay-recommended third-party tools if the troubleshooting or debugging level scope falls outside of the iWay framework.
3. Identify scenarios where the appropriate tool or facility can be best used to troubleshoot and debug the issues as they occur.

Chapter 2

Using iWay Service Manager Diagnostics, Tracing, and Logging Facilities

This section describes how to use the iWay Service Manager Administration Console to perform diagnostic functions. Information on how to run iWay Service Manager (iSM) in a command shell to enter commands that can assist you during troubleshooting is provided. It also describes how to use the Windows Event Viewer for troubleshooting purposes.

The following topics include instructions on how to configure logging and debugging properties to view the resulting log files in the console. Instructions on how to enable the tracing features of iSM are also provided, in addition to information about the iSM test tools that assist in debugging.

In this chapter:

Running in a Command Shell

You can run iWay Service Manager (iSM) in a command shell to debug and troubleshoot any errors that may occur.

On Windows, the *iwsrv* command starts iSM in a command window for debugging purposes. For reference, the following topic includes the full syntax of the *iwsrv* command.

Syntax: How to Start iWay Service Manager in a Command Window (Windows)

Navigate to the iWay home bin directory. For example, on Windows, if iWay is installed in C:\Program Files\iWay61, go to

```
C:\Program Files\iWay61\bin
```

The syntax for the *iwsrv* command, which starts iWay Service Manager in a command window on Windows, is:

```
iwsrv [configuration] [-s service] [-l launch] [options]
```

where:

configuration

Is the name of the server configuration that is loaded for this instance. The default value is base.

service

Is the name of the service that is executed. Valid values are:

start. Starts the server configuration (default).

stop. Stops the server configuration.

install. Installs the server configuration.

remove. Removes the server configuration.

query. Queries the server configuration.

launch

Specifies the launch method. Valid methods are:

java. Loads Java in a separate process and uses the JVM options, NT dependencies, and other preferences found within the iSM configuration that are configured through the console. For example:

```
iwsrv.exe base -s start -l java
```

script file. Specifies a script file that defines the run-time preferences. This script file must be located in the iWay Service Manager installation directory. For example:

```
iwsrv.exe base -s start -l iWay61.cmd
```

Both of the above uses of -l will force the service to load Java in a separate process. When the service is stopped, both iwsrv.exe and java.exe are terminated.

options

Specifies tracing or server back-up information. Valid values include:

-b. Indicates that Service Manager is a back-up server, for example:

```
iwsrv.exe base -s start -b
```

-c. Turns tracing on. In this mode, you can display useful error messages on the console. For example, you can display a message that says the Java Runtime Environment (JRE) is not properly installed. For example:

```
iwsrv.exe base -s start -c
```

-d. Limits tracing to debug only, for example:

```
iwsrv.exe base -s start -d
```

-f. [PATH] filters the system path when invoking JAVA. [RESTART] suppresses the JVM fault restart capability.

-h. iWay61 home directory.

-t. The amount of time (in seconds) to process service shutdown.

Example: Starting a Server Configuration With Traces Enabled

The following command starts a server configuration named *test* and sends traces to the command window as print lines:

```
iwsrv test -c
```

Diagnostic Commands

When iWay Service Manager (iSM) is running in a command shell, you can control it by typing commands in addition to using the console. These commands are designed to assist you in resolving issues.

Several of the key commands are listed and described in the following table. To see a full list of available commands, type *Help* at the Command Prompt on Windows after using the iwsrv command to start iSM.

Command	Description
diagzip	Creates a diagnostic information file for use by iWay Support. For example, you can enter the following command: `diagzip c:\temp\Diag_from_base`
errors	Displays the last ten errors reported by the server.
exits	Displays loaded exits, such as Activity Log and Correlation Manager.

Command	Description
func	Displays the list of iWay Functional Language (iFL) functions, or the parameters of that function.
gc	Runs the Java garbage collector.
help	Displays help for the diagnostic commands. Type the following to display help for a specific diagnostic command: `help command_name`
info	Displays channel information.
license	Displays available iWay license codes.
line	Prints one or more lines on the command window or the trace log to improve the readability.
memory	Lists the amount of memory that is currently in use and the amount of free memory that is available.
pools	Lists resource pools.
providers	Displays providers currently in use.
pull	Load information from another configuration or installation.
quit	Exits the server. All listeners must be stopped.
refresh	Restarts a specified listener with an updated local configuration.
run	Runs a command file.
set	Sets a parameter. Usually used to set the tracing level, for example: `set trace on`
shell	Attempts to run an operating system command.
show	Displays server information.
sregs	Displays globally available special registers.
start	Starts one or more channels.

Command	Description
stats	Run statistics on the current instance or listener.
stop	Stops one or more channels.
threads	Lists execution threads currently controlled by the server. Does not necessarily include threads started by auxiliary packages, such as third-party interfaces. Useful after a Stop command to determine what is still running.
time	Prints the GMT time on the console.
tool	Runs a named tool, such as *testfuncs*.
type	Type or display the contents of a text file.
version	Displays the product version and all later versions of .JAR files.

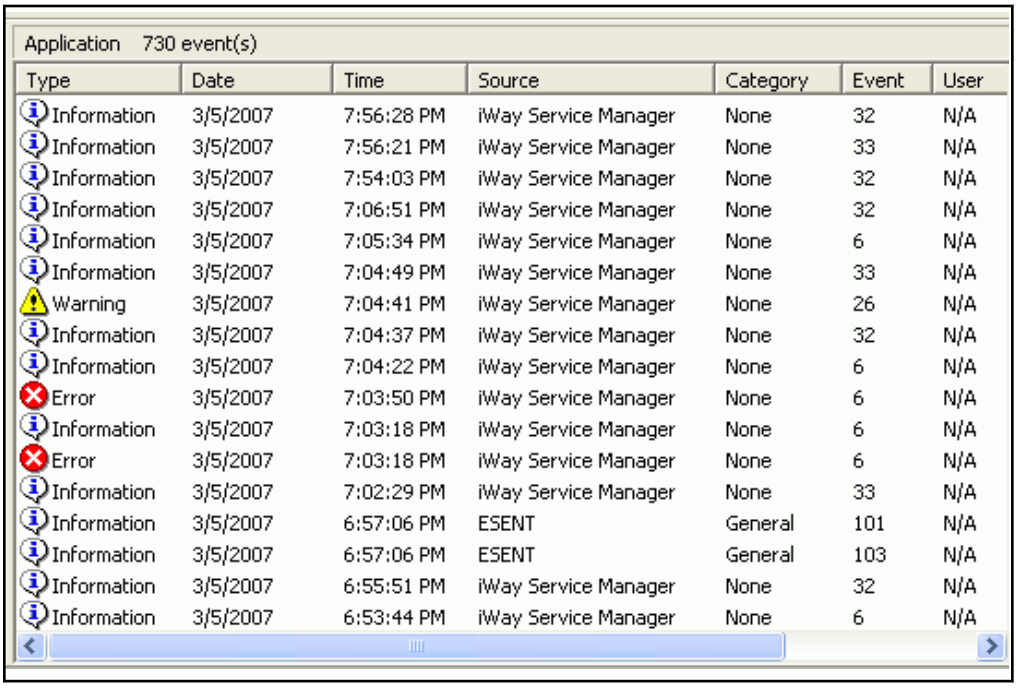
Troubleshooting on Windows

Information, warning, and error messages are logged in the Windows Event Log system. When a problem occurs, the Windows Event Log is the first place to look for information.

Procedure: How to Display Messages in the Windows Event Viewer

1. Access the Event Viewer from Administrative Tools, which can be found in the Windows Control Panel.
2. From the left pane of the Event Viewer, click *Application* to view iWay Service Manager entries.

The following image shows entries in columns that indicate the type of message, the date and time, the source (for example, iWay Service Manager), the category, the event, and the user, when applicable.



Application 730 event(s)

Type	Date	Time	Source	Category	Event	User
Information	3/5/2007	7:56:28 PM	iWay Service Manager	None	32	N/A
Information	3/5/2007	7:56:21 PM	iWay Service Manager	None	33	N/A
Information	3/5/2007	7:54:03 PM	iWay Service Manager	None	32	N/A
Information	3/5/2007	7:06:51 PM	iWay Service Manager	None	32	N/A
Information	3/5/2007	7:05:34 PM	iWay Service Manager	None	6	N/A
Information	3/5/2007	7:04:49 PM	iWay Service Manager	None	33	N/A
Warning	3/5/2007	7:04:41 PM	iWay Service Manager	None	26	N/A
Information	3/5/2007	7:04:37 PM	iWay Service Manager	None	32	N/A
Information	3/5/2007	7:04:22 PM	iWay Service Manager	None	6	N/A
Error	3/5/2007	7:03:50 PM	iWay Service Manager	None	6	N/A
Information	3/5/2007	7:03:18 PM	iWay Service Manager	None	6	N/A
Error	3/5/2007	7:03:18 PM	iWay Service Manager	None	6	N/A
Information	3/5/2007	7:02:29 PM	iWay Service Manager	None	33	N/A
Information	3/5/2007	6:57:06 PM	ESENT	General	101	N/A
Information	3/5/2007	6:57:06 PM	ESENT	General	103	N/A
Information	3/5/2007	6:55:51 PM	iWay Service Manager	None	32	N/A
Information	3/5/2007	6:53:44 PM	iWay Service Manager	None	6	N/A

3. To view its contents, double-click an entry.

   If you are having difficulty starting a service for iWay Service Manager, which cannot be resolved using information from the Event Viewer, start the service in console mode.

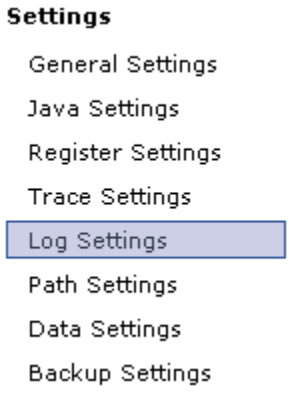
Performing Diagnostic Functions

The iWay Service Manager Administration Console enables you to configure diagnostic properties for logging and tracing. After logging and tracing properties are enabled, you can view the resulting log files in the console.

Log Settings

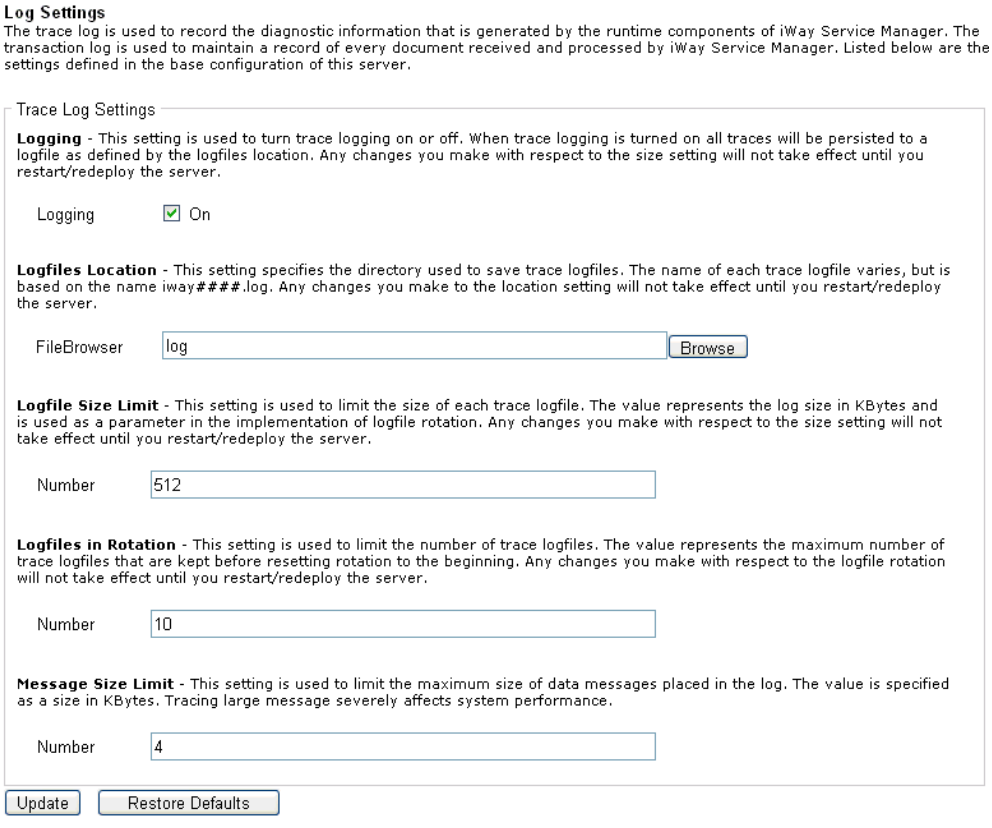
The Trace Log is used to record the diagnostic information that is generated by the run-time components of iWay Service Manager. The Transaction Log is used to maintain a record of every document received and processed by iWay Service Manager. The following procedure describes how to configure log settings that are defined in the base configuration of iWay Service Manager.

Procedure: How to Configure Log Settings



1. In the left console pane of the Server menu, select *Log Setting*s.

The Log Settings pane displays, as shown in the following image.



2. Change the default values.

   For more information, see *Log Setting Properties* on page 14.

3. Click *Update*.

4. For your changes to take effect, restart iWay Service Manager.

Reference: Log Setting Properties

The following table lists and describes the log setting properties.

Property	Description
Trace Log Settings	

Property	Description
Logging	Turns logging on or off. Required if you want to log to a file, use a diagnostic activity log, or view the log online.
Logfiles Location (Directory field)	Directory where the trace log root resides. To create the directory if it does not exist, select the check box.
Logfile Size Limit (Number field)	Maximum allowed for each file size in kilobytes (used for log rotation). iWay recommends a minimum of one megabyte.
Logfiles in Rotation (Number field)	Maximum number of files to keep (used for log rotation).
Message Size Limit (Number field)	Maximum size of the data message in a log file measured in kilobytes. Large trace messages affect system performance.

Trace Settings

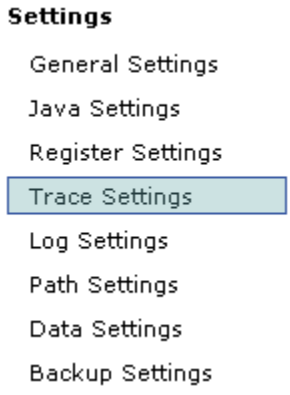
Tracing is key to diagnosing problems and thus to application reliability. iWay Service Manager provides a full complement of tracing services, oriented to diagnostic analysis of the running system. Tracing provides a step-by-step explanation of the internal activity of the server.

It is important to note that tracing can affect system performance. The iWay Service Manager Administration Console enables you to select the levels of traces that you want to generate. Unless you are diagnosing a problem, you should limit tracing to error-level only.

A separate category called JLINK debug masks trace messages originating in the iWay JDBC driver that is used to access the main data server. You can specify actual tracing levels for all instances of the driver in the driver settings of the Data Server Properties configuration window. For more information, see *How to Activate JLINK Tracing* on page 19.

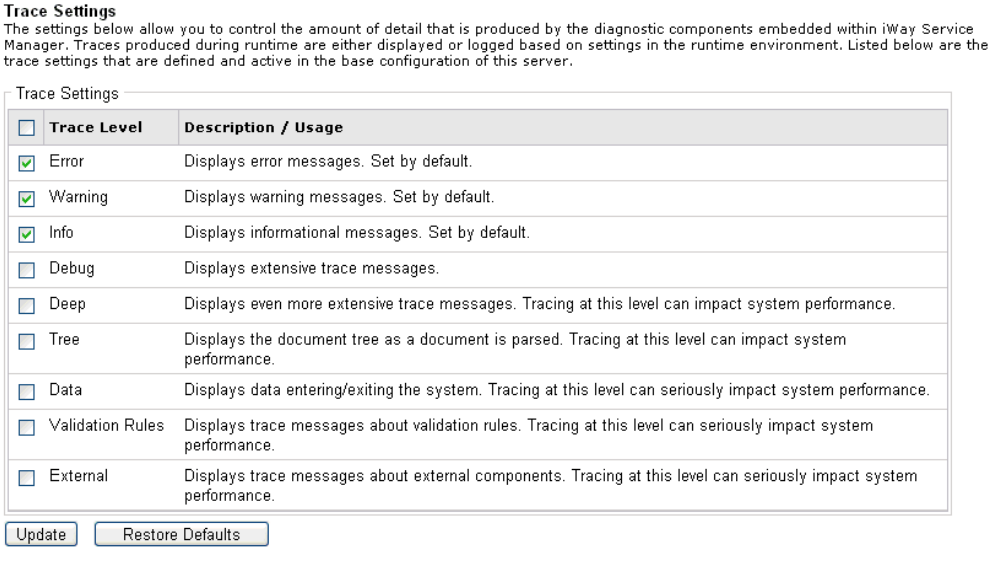
The following procedure describes how to control the amount of detail that is produced by the diagnostic components embedded within iSM. Traces produced during run time are displayed or logged based on settings in the run-time environment.

Procedure: How to Select Trace Levels



1. In the left console pane of the Server menu, select *Trace Settings*.

   The Trace Settings pane displays, as shown in the following image.



2. If other than the default trace levels (Info and Error) are required, select the desired trace level check box.

   For more information, see *Trace Setting Properties* on page 17.

3. Click *Update*.

Reference: Trace Setting Properties

The following table lists and describes the trace setting properties.

Trace Level	Description
Error	Displays error messages. This trace level is set by default.
Warning	Displays warning messages. This trace level is set by default.
Info	Displays informational messages. This trace level is set by default.
Debug	Reports data that is helpful for debugging situations. Shows logic that tracks the path of a document.
Deep	Used for detailed logic tracing. Stack traces are reported by the system in deep debug level. Use only if instructed to do so by iWay Support. **Caution:** Tracing at this level can impact system performance.
Tree	Displays the document as it enters and leaves the system in XML form. This is a level at which intermediate processing as a document evolves is done. **Caution:** Tracing at this level can impact system performance.
Data	Displays the incoming and outgoing documents as they pass to and from the protocol channel. **Caution:** Tracing at this level can impact system performance.
Validation Rules	Displays trace messages about validation rules. **Caution:** Tracing at this level can impact system performance.
External	Displays trace messages about external components. **Caution:** Tracing at this level can impact system performance.

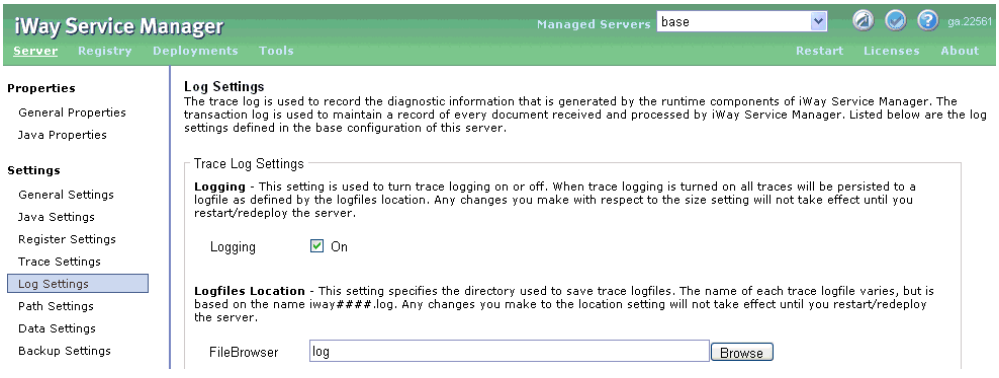
Procedure: How to Log Traces to a File

If tracing is turned on without logging, the tracing information appears only in the debug window and is not saved to a file.

To log traces to a file:

1. In the left console pane of the Server menu, select *Log Settings*.

   The Log Settings pane opens.



2. In the Logfiles Location section, specify the path to the directory used to save log files.
3. Click *Update*.
4. For your changes to take effect, restart iWay Service Manager.

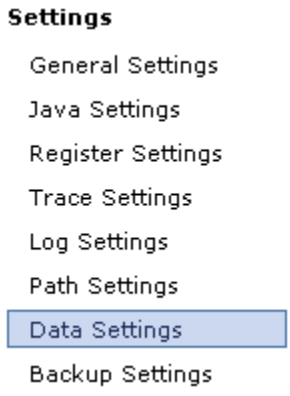
Traces are available in several levels and controlled independently. For more information, see *Trace Setting Properties* on page 17.

Note: Trace settings for managed configurations must be set for each configuration independently.

All levels can be masked, so that the log contains only brief informational and error messages.

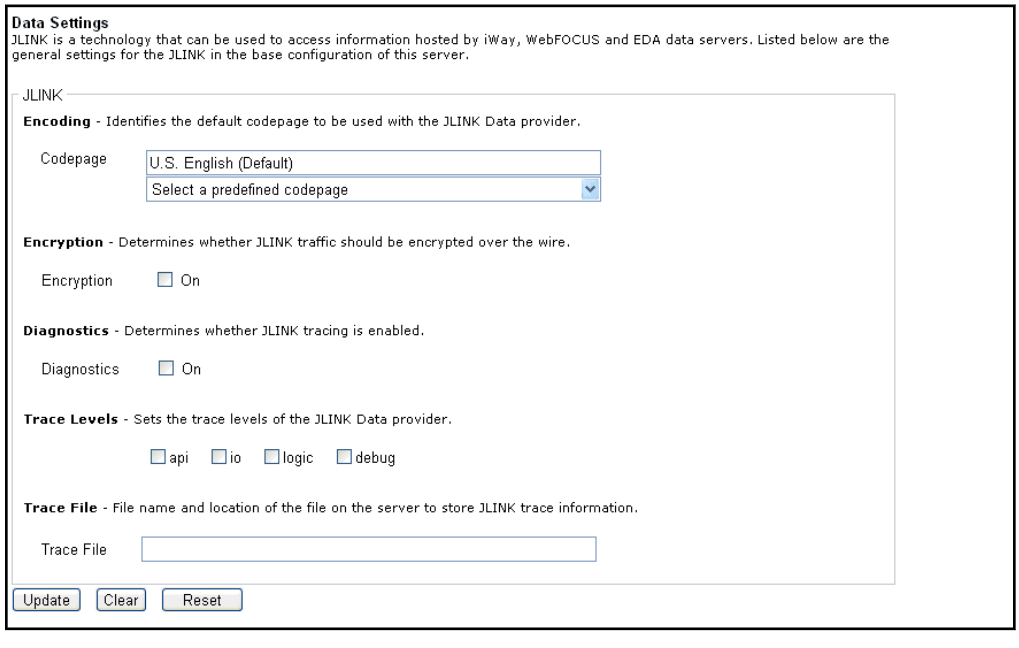
Unlike most design time settings, changing trace levels takes immediate effect in the run-time system. Changing the log file location does not take effect until iWay Service Manager is restarted.

Procedure: How to Activate JLINK Tracing



1. In the left console pane of the Server menu, select *Data Settings*.

   The Data Settings pane opens, as shown in the following image.



   a. Select the *Diagnostics* check box.

   b. Specify trace levels for specific instances of the driver.

      The trace levels are:

      api. Provides entry and exit tracing as the application steps through JDBC calls.

io. Traces data in and out of the system.

logic. Traces the internal activity of the driver. This is equivalent to the Debug trace level of the server.

debug. Traces internal operations of the driver. This is equivalent to the Deep Debug trace level of the server.

c. Type the name of the trace file in the Trace File field.

The trace file specification enables you to route traces from the iWay JDBC driver to a specific file. If you do not specify a trace file, the traces (in most cases) appear in the standard server trace. Certain generalized services that use the iWay JDBC driver do not pass traces through the server. In these cases, specification of the external trace file enables the traces to be captured. You may be prompted to send this file to iWay Support as part of the problem resolution process.

2. Click *Update*.

Measurements and Statistics

The Measurements package allows you to analyze the behavior of iWay Service Manager.

The Statistics package provides the following functionality:

- Reports heap memory usage as an extension to the existing *memory* command.
- Searches for and detects deadlocked workers as part of the extended *threads* command.
- Reports CPU and user time expended by masters as part of the extended *stats* command. Usage statistics can also be sent to an external monitoring facility for more detailed analysis.

In some situations, the Measurements package can add significant overhead to the operation of iWay Service Manager. Therefore, do not use the Measurements package in a production environment unless that environment is undergoing analysis.

The information and formats described in this topic are release-dependent, and subject to change.

Memory

The *memory* command displays the amount of memory in use at the time that the command is issued. When the Statistics package is in use, the standard display is augmented by an additional line that starts with the word Heap.

The Java Virtual Machine has a heap, which is the run-time data area from which all required memory is allocated. The heap is created at the startup of the Java Virtual Machine. Heap memory for objects is reclaimed by an automatic memory management system, which is known as a *garbage collector*. Although the garbage collector runs automatically, you can issue the *gc* command to force it to run for analytic purposes.

The heap memory display has four fields.

Field	Description
init	Represents the initial amount of memory (in bytes) that the Java Virtual Machine requests from the operating system for memory management during startup. The Java Virtual Machine may request additional memory from the operating system, and may also release memory to the system over time. The value of *init* may be undefined (0) on some platforms.
committed	Represents the amount of memory (in bytes) that is guaranteed to be available for use by the Java Virtual Machine. The amount of *committed* memory may change over time (it may increase or decrease). The Java Virtual Machine may release memory to the system, and the value of *committed* could be less than the value of *init*. The value of *committed* is always greater than or equal to the value of *used*.
max	Represents the maximum amount of memory (in bytes) that can be used for memory management. Its value may be undefined. If its value is defined, the maximum amount of memory may change over time. If *max* is defined, the amount of *used* and *committed* memory is always less than or equal to the value of*max*. A memory allocation may fail if it attempts to increase the *used* memory, such that the value of *used* is greater than the value of *committed*, even if the value of *used* is less than or equal to the value of *max* (for example, when the system is low on virtual memory).
used	Represents the amount of memory currently used.

The heap display is more accurate than the memory display issued without the Measurements package installed. The original (standard) information is displayed, in addition to the new heap information.

```
Enter command:>memory
STR00X35: memory used 8244K, free 591K,
   nodes: cache 1001 allocated 8607, reclaimed 116, destroyed 116
   namespace 0 namespace reclaim 0
   Heap: init=0K committed=8244K max=65088K used=7662K

Enter command:>
```

The key value is *used*, which indicates how much memory is currently allocated. As the value of *used* approaches the value of *max*, the garbage collector may start, and performance may be eroded.

Deadlocks

The deadlock detector finds cycles of threads that are in deadlock, waiting to acquire locks. Deadlocked threads are blocked, waiting to enter a synchronization block, or waiting to reenter a synchronization block after a wait call, in which each thread owns one lock while trying to obtain another lock already held by another thread.

A thread is deadlocked if it is part of a cycle in the relation *is waiting for lock owned by*. In the simplest case, thread A is blocked, waiting for a lock owned by thread B, and thread B is blocked, waiting for a lock owned by thread A.

This is an expensive operation. Use it only in cases in which you suspect that messages are locked up in the system.

To enable monitoring, use the command *thread monitor on*. To disable monitoring, use the command *thread monitor off*. While the monitor is enabled, entering the *threads* command displays information regarding deadlocks, such as the thread name or names, and the lock name or names. The thread names indicate the components that are deadlocked.

Statistics

When iWay Service Manager is running without the Measurements package, some statistics are generated with wall clock times. With the Measurements package, the CPU and user state times are also generated. On the summary page, all values for time are reported in seconds, with a precision of four places. It is possible to develop a report with a greater precision for time.

In some cases, wall clock times show useful information. These times provide a measure of performance for a single message as experienced by the sender. They do not provide any information regarding the throughput capacity of iWay Service Manager.

CPU and user times describe the actual execution time expended on messages. Implementation of these measurements depends on the platform and the Java Virtual Machine (JVM). In many cases, the CPU and user times are the same, as the JVM may not discriminate between the two. For those platforms on which the JVM does discriminate, expect the CPU time to be greater than the user time.

User time is the CPU time that the current thread has executed in user mode, that is, the time spent executing iWay Service Manager instructions.

CPU time is the sum of user time and system time. It includes the time spent setting up for JVM services such as locks, network operations, I/O operations, and other services.

```
Enter command:>stats
       In seconds
   name       count      low     high     mean variance  std.dev. ehr num/sec
mq1a       wall:  2  0.0470  0.1560  0.1015   0.0030    0.0545   -    9.85
           cpu :  2  0.0312  0.0625  0.0469   0.0002    0.0156   -
           user:  2  0.0312  0.0625  0.0469   0.0002    0.0156   -
```

The *stats* command displays a summary of statistics gathered up to that point. To reset the values to zero, use the *stats reset* command. iWay recommends that you do not rely on statistics until several messages have been handled to completion, as iWay Service Manager front-loads initialization. Once the system is in a steady state, reset the statistics to zero.

The numbers displayed on the summary page are approximate and are intended for general guidance only. Brief descriptions of the displayed fields are provided in the following table. A fuller understanding of the message processing distribution described here requires some knowledge of statistics and probability, as they apply to queuing.

Field	Description
count	The number of messages that have been handled, for which statistics have been gathered.
low	The lowest time recorded for the handling of a message.
high	The highest time recorded for the handling of a message.
mean	The numeric mean of the times recorded. This value is the sum of the times divided by the number of messages handled. This value is frequently called the average.
variance	The statistical variance of the times recorded. Variance is a measure of how numbers disburse around the mean.

Field	Description
std.dev.	The statistical standard deviation of the times recorded. Standard deviation is a measure of how numbers disburse around the mean.
ehr	The Ehrlang Density Coefficient, which provides evidence of the randomness of the time distribution. If there are too few values to compute the coefficient, a hyphen (-) is displayed. If the coefficient is sufficiently close to constant, the term *const* is displayed. This value is an approximation. A value of 1.0 indicates a Poisson distribution, which is the design point of iWay Service Manager. A very low value can indicate that the individual times recorded are skewed and therefore less usable for predicting behavior.
num/sec	The reciprocal of the mean, providing the number of messages handled per second. This value is displayed for the wall time. It is not a direct measure of the throughput capacity of iWay Service Manager.

The iWay Service Manager Administration Console also displays a summary of statistics. The Listener Statistics pane displays a table similar to the following.

Listener Name	Listener Type	Completed	Time clock	Mean (sec)	Std. Dev. (sec)	Variance (sec)	Ehrlang	Num/Sec
filein	FILE	1	Wall	0.1250	0.0000	0.0000	const	8.00
			CPU	0.1094	0.0000	0.0000	const	9.14
			User	0.0938	0.0000	0.0000	const	10.67
filelock	FILE	0	Wall	0.0000	0.0000	0.0000		
			CPU	0.0000	0.0000	0.0000		
			User	0.0000	0.0000	0.0000		

Emitted Statistics Information

iWay Service Manager can emit statistics as each measurement is generated. Statistics records are included in a comma-delimited file of alphanumeric characters.

The following table describes the fields in the file.

Field	Format	Description
type	String	The value 1, which is the record type. Other record types may be added in a future release.
id	String	The generator (worker) ID.
tid	String	The transaction ID.
msglen	Integer	The message length (non-streaming). If the length cannot be determined, the value -1 is specified.
complexity	Integer	A measure of the complexity of the message. The higher the number, the greater the complexity. This value is generally a measure of the number of nodes in the XML tree. A value of -1 means unknown. For most purposes, the number of digits that this integer has is a good value to analyze.
timestamp	Integer	The current time, in milliseconds. This value is the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC (Universal Time, Coordinated). The value is a timestamp for the record.
gregtime	String	The timestamp in GMT (Greenwich Mean Time). The format is: yyyy-mm-dd**T**hh:mm:ss:mmm**Z**
walltime	Float	The wall clock time expended, in milliseconds.
usertime	Float	The user time expended, in milliseconds.
cputime	Float	The CPU time expended, in milliseconds.
usedmem	Integer	The used memory, in kilobytes (K). See *Memory* on page 20.
committed	Integer	The committed memory, in kilobytes (K). See *Memory* on page 20.

Field	Format	Description
$	String	The end-of-record indicator.

The following is a sample record in the file:

```
1, W.udp1.1, udp1-UDP-W.udp1.1_20050328191546135Z, -1,1735, 1112037346213,
2005-03-28-T19:15:46:135Z, 78.0, 15.625, 15.625,1200,800 $
```

To enable iWay Service Manager to emit these statistics, define the following to the JVM properties

```
-Dstaturl=host:port
```

where *host* and *port* are a UDP receiver.

You can specify the host and port of an iWay Service Manager UDP listener that has a process that is defined to handle incoming messages. Use the iWay Service Manager Administration Console to help define Java system properties.

The iwmeasure.jar extension provides a simple StatsGather agent that appends each record to a named file.

Do not run the statistics gathering component on a machine that is being measured. The process of receiving the statistics will be measured, creating a loop. You must use two configurations, preferably on separate machines.

Tips

- When you work with the complexity of a document (a number greater than -1 in the complexity field), a good guideline is to use the number of digits in the field. For example, a message that consists of 172 nodes would get a complexity measure of 3, while a message that consists of 1459 nodes would get a measure of 4. For most analytic purposes, this provides a reasonable value.
- Traces use the bulk of time and memory in iWay Service Manager. For valid statistics, turn off all traces. You can use the *set trace off* command to do this, or you can use the iWay Service Manager Administration Console.

Many books are available on queuing theory, the use of available statistics, and the interpretation of displayed fields.

Kushner, Harold J.; *Heavy Traffic Analysis of Controlled Queuing and Communications Networks*. New York, Springer; (June 8, 2001).

Using the Log Viewer

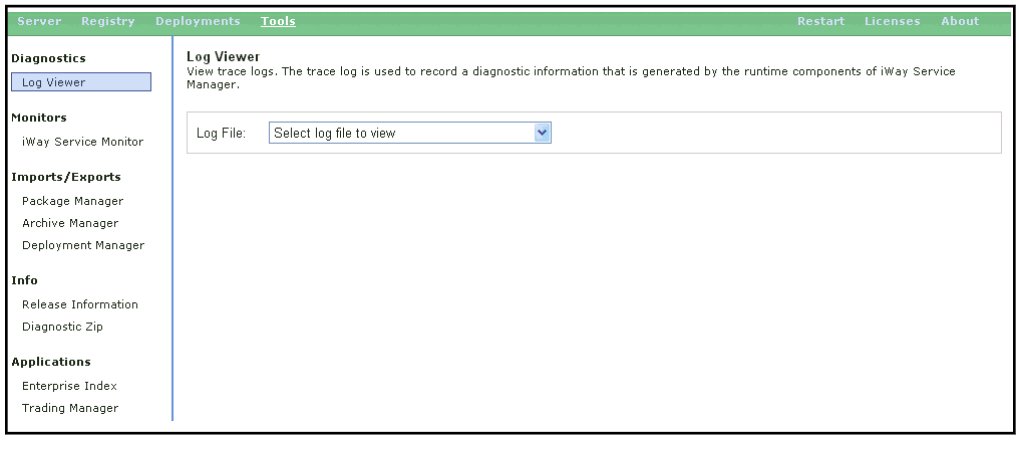
The Log Viewer manages the display properties of system debugging information when the logging and tracing functions are activated. It filters and displays debugging information as each transaction is received and processed. The Log Viewer also displays the date/time range, type, source, and message of every trace entry.

Note: In order to display traces of a specific level, you must have previously enabled them to be written to the log file. For more information, see *Log Settings* on page 12 and *Trace Settings* on page 15.

Procedure: How to Use the Log Viewer

1. Click *Tools* in the top pane and select *Log Viewer* from the Diagnostics section in the left pane.

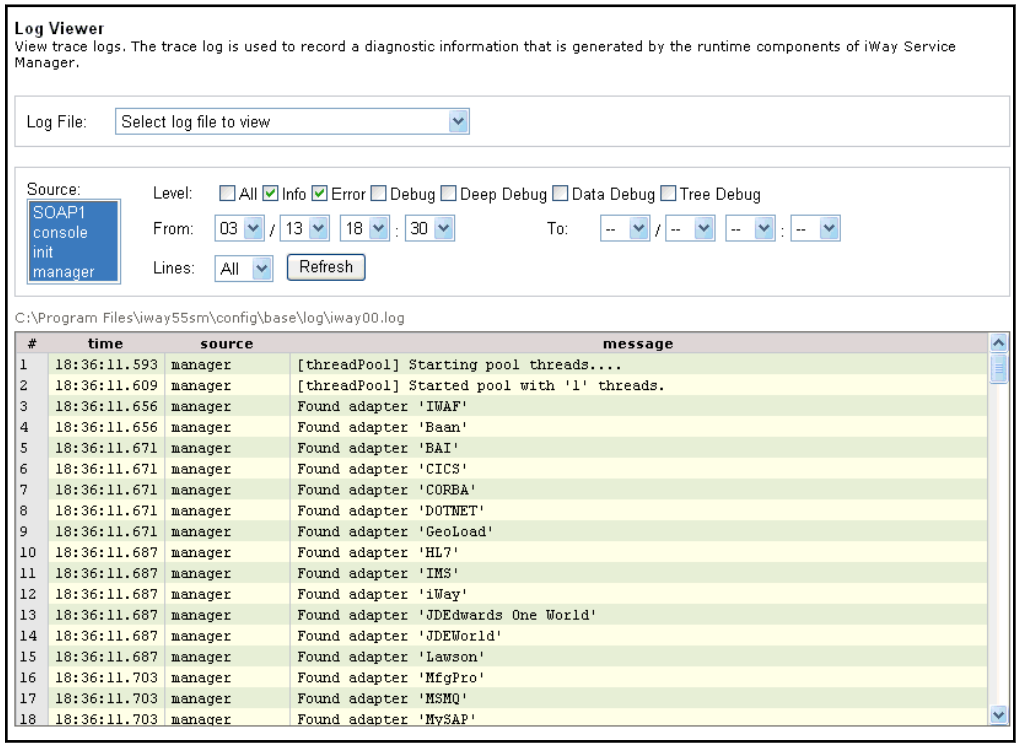
   The Log Viewer pane opens, as shown in the following image.



2. Select a specific log file to view from the Log File drop-down list.

   Note: Log file names are reused in a circular queue so that they will not proliferate and consume too much disk space. The date and time stamp is shown in the drop-down list in order to show the correct sequence of the files.

The Log Viewer pane is automatically refreshed and shows the log file you selected.



#	time	source	message
1	18:36:11.593	manager	[threadPool] Starting pool threads....
2	18:36:11.609	manager	[threadPool] Started pool with '1' threads.
3	18:36:11.656	manager	Found adapter 'IWAF'
4	18:36:11.656	manager	Found adapter 'Baan'
5	18:36:11.671	manager	Found adapter 'BAI'
6	18:36:11.671	manager	Found adapter 'CICS'
7	18:36:11.671	manager	Found adapter 'CORBA'
8	18:36:11.671	manager	Found adapter 'DOTNET'
9	18:36:11.671	manager	Found adapter 'GeoLoad'
10	18:36:11.687	manager	Found adapter 'HL7'
11	18:36:11.687	manager	Found adapter 'IMS'
12	18:36:11.687	manager	Found adapter 'iWay'
13	18:36:11.687	manager	Found adapter 'JDEdwards One World'
14	18:36:11.687	manager	Found adapter 'JDEWorld'
15	18:36:11.687	manager	Found adapter 'Lawson'
16	18:36:11.703	manager	Found adapter 'MfgPro'
17	18:36:11.703	manager	Found adapter 'MSMQ'
18	18:36:11.703	manager	Found adapter 'MySAP'

3. Select the source component, level, date and time range, and number of lines to display and click *Refresh*.

   The contents of the log file, as filtered by your criteria, are displayed.

   Tip: Multiple trace sources can be selected by pressing Ctrl and clicking the trace source.

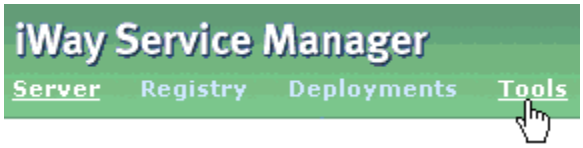
Creating a Diagnostic Zip

The Create Diagnostic Zip option provides a quick way to collect the current configuration and log files. An iWay Software support representative may ask you to create a diagnostic zip for problem analysis.

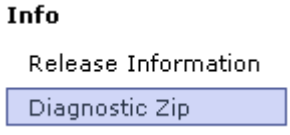
You can use this function to add any relevant comments to the file. The file is labeled with a timestamp in your configuration directory.

Note: Remove previous trace files prior to running a diagnostic zip.

Procedure: How to Create a Diagnostic Zip

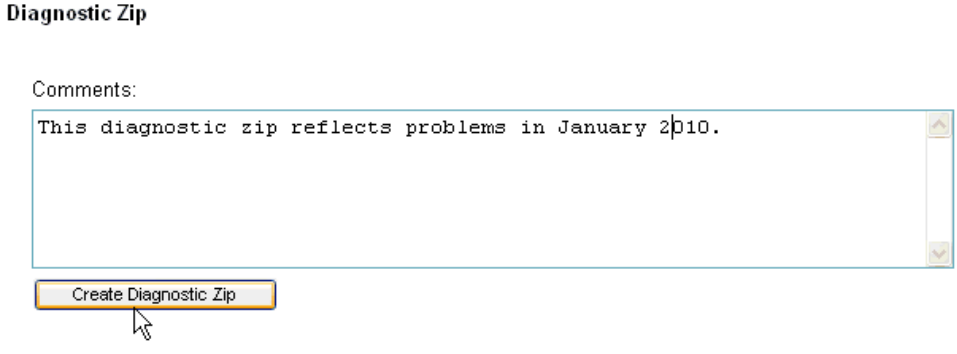


1. Click *Tools* in the menu bar, which is located in the top pane.



2. In the left pane, select *Diagnostic Zip*.

   The Diagnostic Zip pane opens, as shown in the following image.



3. Type your comments in the space that is provided.
4. Click *Create Diagnostic Zip*.

   In this example, if you are using the base configuration, the file is saved to the location shown in the following image.

Chapter 3

Identifying Available Services for Troubleshooting and Debugging

This section identifies the services (agents) that are provided by iWay Service Manager (iSM) for troubleshooting and debugging purposes.

In this chapter:

Activity Log Entry Service (com.ibi.agents.XDXALogEvent)

Syntax:

```
com.ibi.agents.XDXALogEvent
```

Description:

This service is used to record events to the system log during a process flow. It can record security events and arbitrary user event codes. Each message that is logged has a type, code, and optional message.

Parameters:

Parameter	Description
Transactional	Determines whether this event record will be included in the log based on the transaction. If set to *true*, the event is logged only if the entire process flow is successful. For transactional recording, the channel must be declared to control the local transaction.

Parameter	Description
Event	The event class to be included in the log. Select one of the following values from the drop-down list: ❑ emit {emit} ❑ security {security} (default) ❑ emit {signature} ❑ crypto {crypto} ❑ user {user} You can also type an arbitrary, user-defined value in the Event field that must be greater than 1000.
Code	A code that further describes the event. Select one of the following values from the drop-down list: ❑ start {start} (default) ❑ end {end} ❑ fail {fail} ❑ sign {sign} ❑ encrypt {encrypt} ❑ decrypt {decrypt} ❑ verify {verify} You can also type an arbitrary, user-defined value in the Code field that must be greater than 1000.
Message	An arbitrary message that you want to associate with this event record.

Available Response Edges for XDXALogEvent

When you connect the XDXALogEvent object to an End object using the OnCustom build relation in a process flow, the available line edges are provided in the Line Configuration dialog box.

The following table lists and describes the available line edges for the XDXALogEvent object.

Line Edge	Description
OnError	Error
OnSuccess	Success
OnFailure	Failure

Catch Service (com.bi.agents.XDCatchAgent)

Syntax:

```
com.bi.agents.XDCatchAgent
```

Description:

Error handling in iWay Service Manager process flows can be accomplished in a number of different ways. The possible methods are:

- Explicitly checking for an error, post-service execution, by conditioning the edge with onError or onFailure.
- Including an outlet conditioned with _iserror().
- Including XDCatchAgent at the beginning of the channel. This channel has two edges on the output side that are used for processing. The first is the onCompletion edge. The second is the onCustom edge, with the onError and onFailure cases selected.

The concept of the XDCatchAgent is similar to a try-catch block in other programming languages.

In other programming languages, a block of code is enclosed between the braces of a try statement. Following the try block is a catch block of code that is enclosed in braces. The code in the catch block has statements that handle any errors that might occur in the try block.

When the thread of execution starts, each line in the try block of code is executed. If each statement is successful, execution continues at the statement following the closing brace of the catch block (assuming that there is not a finally block). If an error occurs within the try block, the thread of execution jumps to the code inside the catch block.

In an iWay Service Manager flow, you can add an XDCatchAgent in front of the services in which an error might occur. There are two edges off this service:

- onCompletion (blue)
- onCustom (brown)

The completion edge is the thread of execution in which everything works in a perfect scenario. All the edges after the service connected by the onCompletion edge are then connected to the onSuccess edge.

The onCustom edge has three selected cases (onError, onFailure, and error_retry). Any errors or failures that occur within the path of the process flow are directed down the onError and onFailure edge. The logic in this branch contains any services necessary to handle errors. The error_retry edge is followed when there is a retry exception. For example, when a SQL Object contains an invalid URL in the process flow, the onCustom/error_retry edge will be followed.

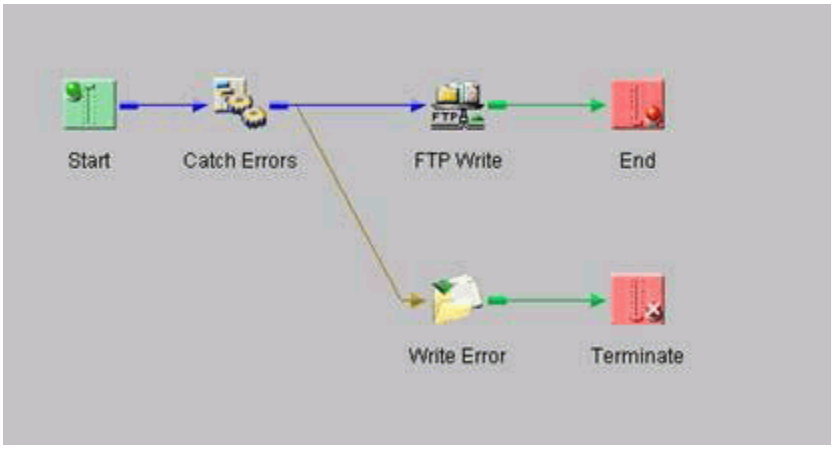
Think of the onCompletion path as the try block and the onCustom edge as the catch block.

You can add multiple XDCatchAgents into a process flow. The error branch is taken off the closest XDCatchAgent previous to where the error occurred. In this manner, you can add multiple error conditions for a given process flow if needed.

Example:

In this example, a file is put into a directory after its creation from a previous channel. The sample process flow is responsible for transmitting the file to the customer FTP site.

Since this is an FTP site, it is subject to network and site availability and other possible outside issues. An error handling strategy is required so that none of the documents being processed are lost because of an outside issue.



In process flow, the XDCatchAgent immediately follows the Start block. An onCompletion edge connects the Catch Errors block to the FTP Write block. The FTP Write block is an FTP emitter that is set up to write the file to an FTP site. The service directly following the XDCatchAgent (Catch Errors) must have an onCompletion edge for this to work correctly.

Following the FTP Write block is the End block. The edge connecting these two services is an onSuccess edge. If a different edge were used and an error occurred, the error edge off of Catch Errors may not be executed.

The onCustom edge of Catch Errors has the onError and onFailure cases selected for the properties. This edge leads to a file write service, Write Error, that puts the file into a hold directory for later reprocessing. Following Write Error, there is an End with a Terminate since no further processing is required at this point. In a real world scenario, a requirement might be that an email is sent if the site is down.

When the target FTP site is up and available, the files are written to the FTP site. If the FTP site is down or you cannot connect to it, the FTP write service will generate an error. This error causes the next execution point to be the File Write to save the file for further processing.

Fail Service (com.ibi.agents.XDFailAgent)

Syntax:

`com.ibi.agents.XDFailAgent`

Description:

The failure business service always returns an XDException. If the retry option is selected for the Type of failure parameter, the exception calls for a retry of the input, if possible. This service is useful when debugging rollback logic in a customer business service.

Parameters:

Parameter	Description
Type of failure	The type of failure to be thrown. Select one of the following options from the drop-down list: ❑ fail ❑ retry
Message	Message to be issued to the user.

Parameter	Description
Bypass Error Message	Indicates not to trace at the error level when this service terminates the process. This is useful for cases where the trace log is being monitored by an external program for errors. The termination will be traced at the debug level.
Call at EOS?	In a streaming environment, EOS (End of Stream) is the short message that is sent after the last document, which signifies the EOS. This parameter determines whether this service should be called for the EOS message. The default value is false.

Example:

This service can be used in situations where a failure must be reported or simulated (for example, if a certain fatal condition is reported). If you run this service in a process flow using <test/> as the incoming message and the retry option is selected for the Type of failure parameter, the following document is returned:

```
<?xml version="1.0" encoding="UTF-8" ?>
<eda>
  <error code="6" timestamp="2009-06-05T19:56:59Z"
  source="com.ibi.agents.XDFailAgent" stage="AGENT">XD[RETRY] cause: 0
  subcause: 0 message: Retry requested from XDFailAgent</error>
</eda>
```

Otherwise, a retry is silent and the incoming message is retried later.

If the fail option is selected for the Type of failure parameter, the process flow handles the termination as an error condition by searching upward on the execution edge to locate a catch node. If none is found, then the process flow is terminated. If a catch node is found, then standard catch logic is performed.

It is not recommended to design process flows in which failure indications are used to control execution logic (for example, simulating a long jump). Failure indications should only be used in the event of actual failures.

QA Service (com.ibi.agents.XDQAAgent)

Syntax:

```
com.ibi.agents.XDQAAgent
```

Description:

This service emits a flattened copy of the input document to a file named in the init() parameters. The service outputs the document (XML or flat) in QA, ondebug, or always modes, depending on the configuration setting of the When parameter. If the QA mode is not enabled (in the Diagnostic System Properties Console Configuration page or by using the set command) and the *always* option is not set, then this service functions as a move service. This service is designed to work as a chained service for debugging purposes. The document and all special registers are included in the output.

The QA mode for iSM can be set by executing the following command to enable the QA mode for the configuration:

```
set qa on [-save]
```

To deactivate the QA mode, execute the following command:

```
set qa off [-save]
```

The QA mode must be enabled for the iSM configuration you are using in order for the QA service to output documents when set to QA mode.

To enable the QA service to output documents on debug, set the iSM debug special register to true. To deactivate the debug mode, set the debug special register to false.

Parameters:

Parameter	Description
Where *	File pattern to receive trace file.
When	Determines when to emit the information. Select one of the following options from the drop-down list: ❏ qa (default) ❏ always ❏ ondebug
Name	Identifier name to mark emitted trace document.
Emit input	Location (file pattern} to which to emit actual input document. If omitted or empty, the incoming document is not emitted.
Base64 Decode	If set to *true*, the value is assumed to be in base64 notation. Only applicable when a specific write value is specified.

Parameter	Description
Starting Offset	If set, this value represents the starting offset within the data block to start the dump.
Maximum Length	If set, this value represents the total number of bytes to dump. If not set, the dump starts from the value specified for the Starting Offset parameter to the end of the buffer.

Trace Message Writer Service (com.ibi.agents.XDTraceAgent)

Syntax:

```
com.ibi.agents.XDTraceAgent
```

Description:

This service writes a message to the trace log of the system. The trace log accumulates messages to record the progress of activity within the server. Usually the trace log is used for debugging purposes.

Parameters:

Parameter	Description
Trace Level	The trace level at which the message is written to the trace log. Select one of the following trace levels from the drop-down list: ❑ Error ❑ Warn ❑ Debug
Message	The message to be written to the trace log.
Call at EOS?	In a streaming environment, EOS (End of Stream) is the short message that is sent after the last document, which signifies the EOS. This parameter determines whether this service should be called for the EOS message.

The edges returned are listed in the following table.

Edge	Description
success	The line was successfully sent to the trace system.
fail_parse	iFL used in the parameters was invalid.

The trace message is written to the trace log at the error or debug level. The trace system must be configured to accept the message on the issued level. Users must be made aware that the use of a trace log can adversely affect server performance.

Caution: The trace log is not the transaction log, which can hold messages regarding operations within the server. Messages can be written to the transaction log using the Activity Log Entry service (XDXALogEvent) and the Activity Log Business Error Message service (XDXALogBizErr). While trace messages are usually free format and designed to help debug a problem, the XALog (transaction log) is more structured and often has externally imposed security, event code, and format constraints.

Chapter 4

Identifying Available Commands and Functions for Troubleshooting and Debugging

This section identifies the commands and functions that are provided by iWay Service Manager (iSM) for troubleshooting and debugging purposes.

In this chapter:

Using the Testfuncs Tool

The *testfuncs* tool enables you to test an expression in the iWay Functional Language (iFL). It evaluates the expression and returns a result. The *testfuncs* tool is intended for technical users. To use the *testfuncs* tool, type the following command:

```
Tool testfuncs <path to
an xml document>
```

This tool supports the *set* subcommand to set a Special Register (SREG) value. The following example shows how to test a SREG with arithmetic:

```
funcs->set aa 1
stored
funcs->sreg(aa)+2
<superroot [baseNode]>
   <arith [funcNodeMath]>+
      <sreg [funcNodeFunctionSreg]>sreg(aa)
         <p-literal [funcNodeLit]>aa</p-literal>
      </sreg>
      <x-literal [funcNodeLit]>2</x-literal>
   </arith>
</superroot>
3
funcs->
```

The following example may be more easier to follow:

```
funcs->_substr('abcde',2,4)
<superroot [baseNode]>
   <substr [funcNodeFunctionStr.substr]>_substr(&apos;abcde&apos;,2,4)
      <p-literal [funcNodeLit]>abcde</p-literal>
      <p-literal [funcNodeLit]>2</p-literal>
      <p-literal [funcNodeLit]>4</p-literal>
   </substr>
</superroot>
cd
funcs->
```

Each test shows the abstract syntax tree that results from when the function is compiled. Problems with the compilation can usually be understood by analysis of the tree. Some function testing requires use of SREGs. The following command sets the specified SREG to the designated value:

```
set regname value
```

The value operand is evaluated, so that a value can, for example, reside in a file. If the value after evaluation begins with a left bracket, the test tool assumes that the value is to be parsed as XML and an XML tree is to be loaded into the named register. Otherwise, the register is set to a string of the input value. To set register one to the value valone, use:

```
funcs->set one valone
stored
```

A file in the root named sregdoc.xml contains an XML document. To load it into a SREG named xmldoc, use the following command:

```
funcs->set xmldoc file('/sregdoc.xml')
stored xml
```

Using the Testxpath Tool

The *testxpath* tool enables you to test XPath expressions against a standard document. The XPath expression will be evaluated against the provided document and the result returned. To use the *testxpath* tool, type the following command:

```
Tool testxpath <sampledocument>
```

Assume that the standard document is the same one as shown in the testfuncs tool example:

```
Enter command:>tool testxpath \smalldoc.xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<a>
   <top>
      <b>one</b>
      <b>two</b>
      <x>sreg(iwayhome)</x>
   </top>
</a>
```

Now enter an XPath expression against the sample document:

```
xpath->/a/top/b[1]
<superRoot [baseNode]>
   <a [baseNode]/>
   <top [baseNode]/>
   <b [baseNode]>
      <functionPredicate [filterInt]>1</functionPredicate>
   </b>
</superRoot>
values->'one'
tree-><?xml version="1.0" encoding="ISO-8859-1" ?>
<xpathresult>
   <b>one</b>
</xpathresult>
list-><b>one [parent=top]
xpath->
```

Using the Flow Command

Runs a previously published process flow. The process flow is run under control of the server configuration, rather than under control of a channel. Channel services may not be available within the process flow. The flow command can be used to test process flows, including verifying that the process flow produces the expected result.

To issue a flow command, enter the following:

```
flow <flowname> [<input> [-x | -f]] [-c] [-o [@outfile]] [-map pairs…]
```

where:

`flowname`

Is the name of the flow. The flow must have been published to the system area of the configuration under which it is to be run.

`input`

Is the input to be supplied to the flow. The input can be in flat form (not parsed into XML) or in XML. The -x (default) or -f switches set the type. If the input is flat it will be passed as a string in Unicode, and not in byte form. Input specification is subject to iFL operation. If the input is omitted, a standard signal message will be passed to the flow. You can use the _file() iFL function to load the contents of a file to be passed to the flow.

`outfile`

Is the path to a file into which the output document contents are flattened. Use of this feature requires that the -o switch is used.

`-c`

Runs the flow transactionally. If this switch is omitted, the flow is not run under transactional control.

`-o`

Is the output of the flow is displayed in the log. If this is omitted, the output is not displayed. The output is the contents of the documents that are sent to the end nodes of the flow.

`-map pairs`

Adds token=value pairs to the standard signal document if used, as the parameter map. The pairs will also be set as DOC level special registers in the execution environment. This must be the last switch on the line, and all tokens that follow it are considered as token=value pairs. The equal sign (=) and commas are optional.

Example: Run a published flow named status.mail. Pass in the name of the channel to monitor. The flow must look in the standard signal document to get the channel name to monitor. The details of the flow are not shown here.

```
flow status.mail -map channel chan1
```

Because no input was specified in the command, a standard signal document will be passed to the flow. It will look like:

```
<signal  type='flow' timestamp='time'  version='2' protocol='command'>
    <parms count='1'>
        <parm name='channel>chan1</parm>
    </parms>
</signal>
```

If the optional -expects switch is used, the flow result is compared on a character-by-character basis with the contents of a named file. If the result of the flow matches the expected result, the following command is emitted to the output trace:

```
match
```

If the two do not match, the flow emits the following command:

```
nomatch
```

In this case, information showing the location of the mismatch and what was found is traced. For example, consider a regression test of the flow “passthru”.

```
Enter command:>flow passthru _file(c:/docs/flowin.xml) -expect c:/docs/
expect.xml
Flow 'passthru' OK, not committed
Unequal compare:
-- Lengths are not equal
Length expected=54, actual=53
Difference starts at char 46,  expected=c'b'/d'98', actual=c'<'/d'60'
Partial Expected: >aaaab</Test>
Partial Actual: >aaaa</Test>
Expected: <?xmlversion='1.0'encoding='UTF-8'?><Test>aaaab</Test>
Actual  : <?xmlversion='1.0'encoding='UTF-8'?><Test>aaaa</Test>
nomatch
Enter command:>
```

Using the Line Command

Prints one or more lines on the command window or the trace log to improve the readability. This is useful as an eye catcher when you are reading a long trace file or command log file.

The line command uses the following format:

```
line [<count>] [-log]
```

where:

<count>

Specifies the number of lines to print. The default is one.

-log

Writes the specified number of lines to the trace log instead of the command window.

For example:

```
Enter command:>line 2

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

-----------------------------------------------
```

Using the _eval() Function

The _eval() function evaluates a string as an expression of the function. This function uses the following format:

```
_eval(expression [,tracemsg [,level]])
```

Property	Type	Description
expression	string	The string to be evaluated.
tracemsg	string	A trace message to be issued when the expression is evaluated.
level	keyword	The trace level specified for the tracemsg attribute. The following trace levels are supported: ❑ **none.** Does not return any traces. ❑ **error.** This setting provides error level traces. ❑ **debug.** This setting provides debug level traces (default). ❑ **deep.** This setting provides deep level traces.

A common use of the _eval() function is to store a complex expression in a file. The expression can be used by _eval(_file(<path>)). Assume that the file /myfilescfg.txt contains the simple expression _sreg('iway.config','none'). If the _file('/myfilescfg.txt') function is used alone, then the value will be _sreg('iway.config','none'), the value in the file. However, by using the _eval() function, the _sreg() is evaluated and the result is the name of the configuration in which the server is running.

An optional tracing service adds a message at the specified trace level (if enabled) to the current trace log. This is useful for debugging the value to be output by this function. By including the special token (%v) in the trace message, the expression value can be included in the message. For example:

```
_eval('file(/holdifl.txt)','eval got %v','deep')
```

Chapter 5

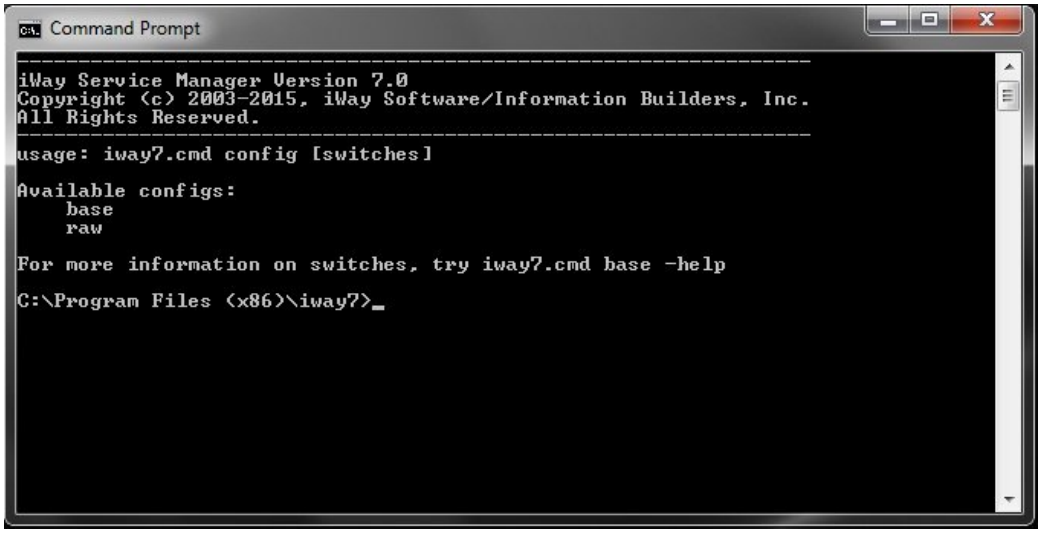
Creating and Using a Remote Command Console

This section describes how to create and use a remote command console in iWay Service Manager (iSM).

In this chapter:

Remote Command Console Overview

iWay Service Manager (iSM) commands such as *start* or *flow* can be entered at the original command window if iSM (the server) is started as a task with a visible window (for example, starting from a command line such as iway7.cmd).



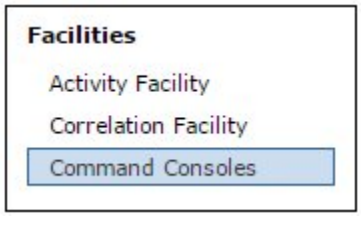
Additionally, commands can be entered using a remote command facility using Telnet (with or without Secure Sockets Layer (SSL)) or Secure Shell (SSH). In either case, the full set of iSM commands is available to the user, depending on the security level at which the logged in user has been granted.

A remote command channel is configured by a configuration console user, and need not be part of a deployed iWay Integration Application (iIA) or configuration until it is required.

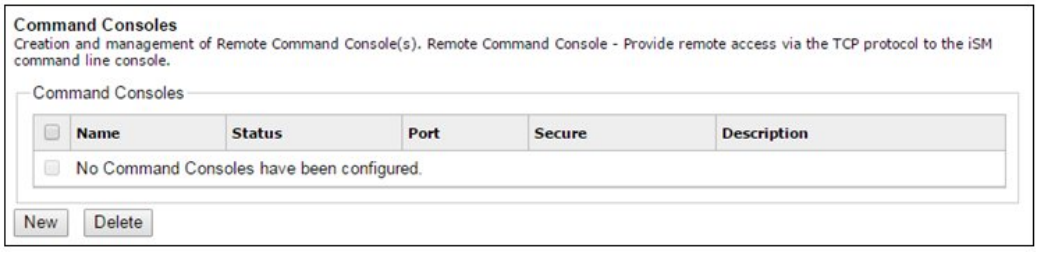
Usually the remote command channel runs off of the base configuration, and the remote command is used to address other running configurations either on the same or another host. A remote command console can be configured to any configuration that is currently running on a host.

Creating a Remote Command Console

The remote command console is created and managed as a facility in the standard iSM Administration Console. To create a new remote command console, click *Command Consoles* in the Facilities group on the left pane, as shown in the following image.

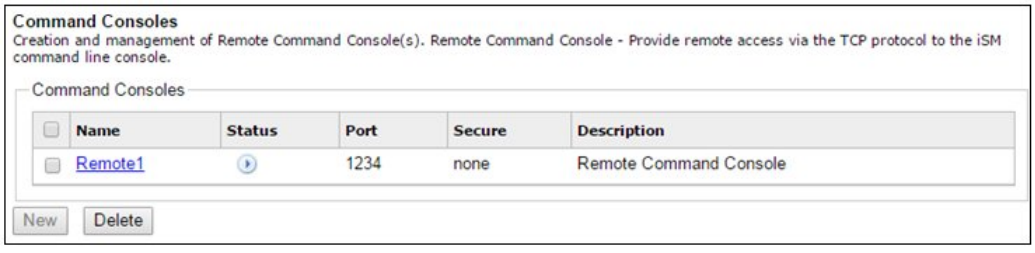


The Command Consoles pane opens, as shown in the following image.



If no remote command consoles have been configured, then the screen will be empty, as currently shown.

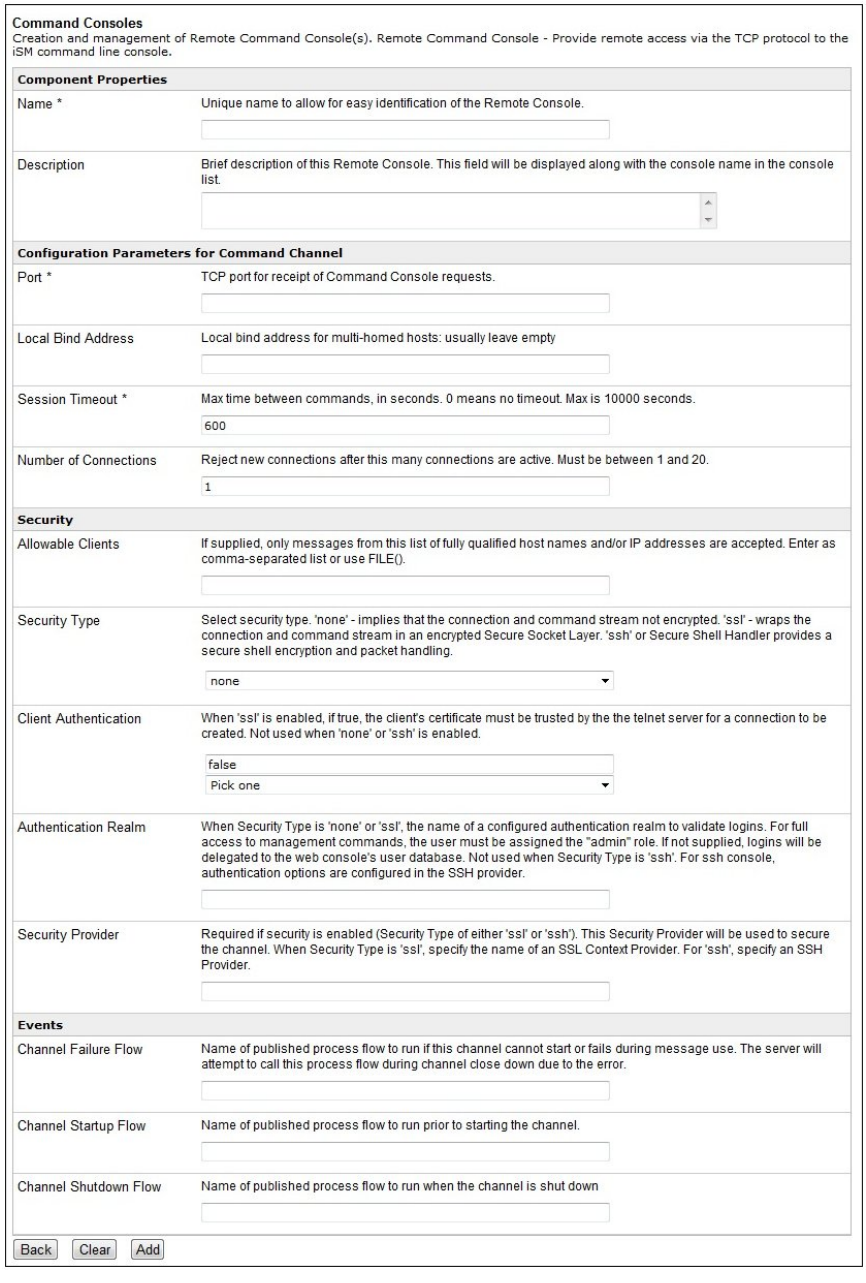
If a remote command console has been configured, then it will be listed in the Command Consoles pane (for example, *Remote1*), as shown in the following image.



Note: You can only have a single remote command console configured in any given configuration.

Click *New* in the Command Consoles pane to configure a remote command console.

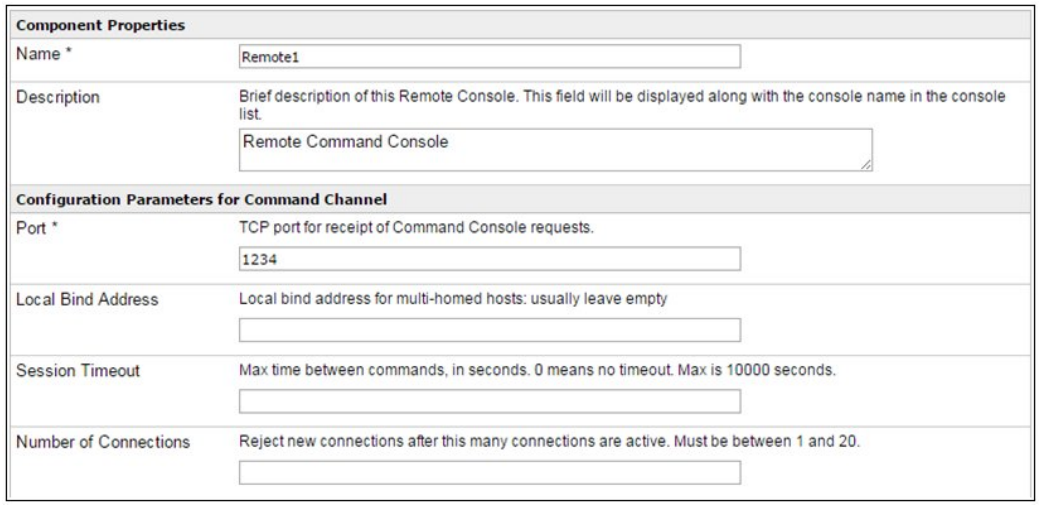
The Command Consoles configuration pane opens, as shown in the following image.



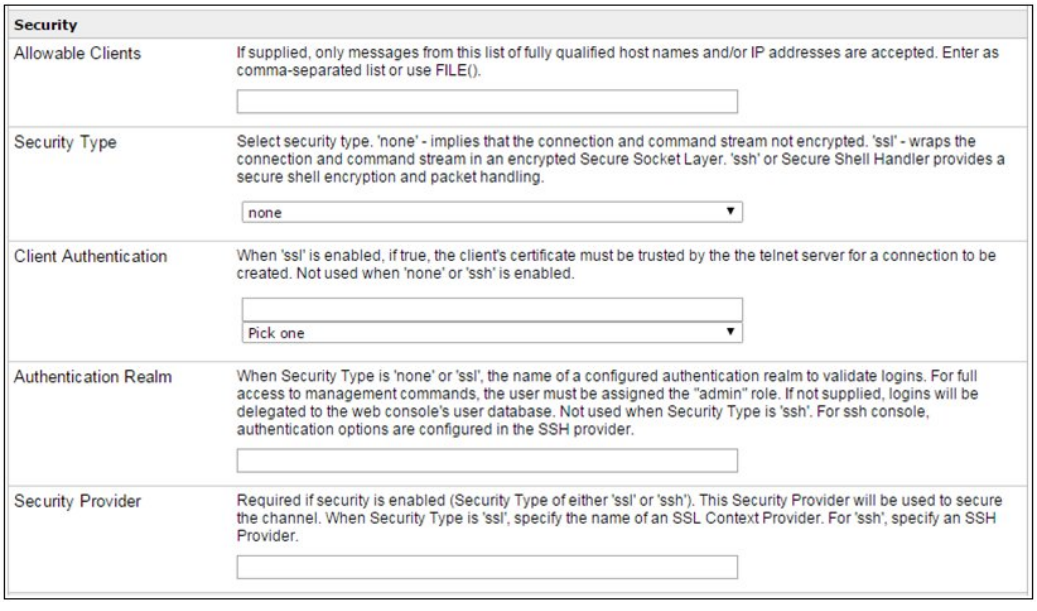
The Command Consoles configuration pane contains a table with the following groups of parameters:

- **Component Properties.** Name and description of the *listener*. This name appears in some logs.
- **Configuration Parameters for Command Console.** Basic parameters including port, sessions, and so on.
- **Security.** Security definitions for the remote command console.
- **Events.** Event-handling parameters that can be configured to run specific process flows when the channel fails, starts, or is shut down.

The first groups (Component Properties and Configuration Parameters for Command Console) define the remote command console and how it will be reached. If no other parameters are configured, then the remote command console will be a standard Telnet command console using the console realm for security.

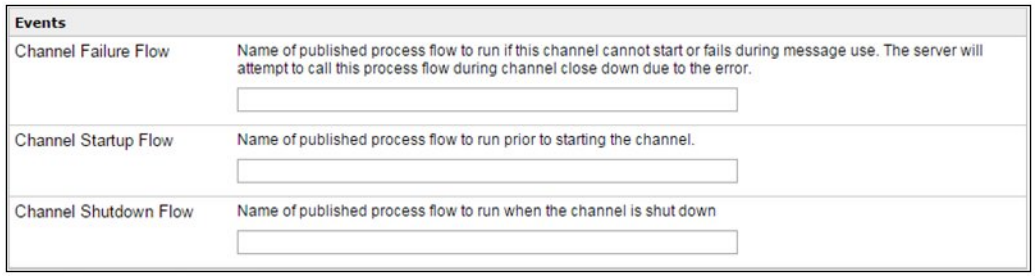


The Security group can be configured as needed. In this case the remote command console will operate using SSH, with a configured realm (for example, LDAP) and an underlying SSH provider. For more information, see the *iWay Service Manager Security Guide*.



Parameter	Applies to Telnet?	Applies to Telnet SSL?	Applies to SSH?
Allowable Clients	Yes	Yes	Yes
Security Type	N/A	N/A	N/A
Client Authentication	No	Yes	No
Authentication Realm	Yes	Yes	No
Security Provider	No	Yes (SSL provider)	Yes (SSH provider)

Events are supported in the Events group, as shown in the following image.



The following table lists and describes each of the available configuration parameters for a remote command console.

Note: An asterisk indicates a required parameter.

Parameter	Definition
Component Properties	
Name*	A unique name that will be used to identify the remote command console.
Description	A brief description for the remote command console, which will also be displayed in the Command Consoles pane.
Configuration Parameters for Command Console	
Port*	TCP port for receipt of Command Console requests.
Local Bind Address	Local bind address for multi-homed hosts: usually leave empty
Session Timeout*	The maximum time between commands, in seconds. A value of zero (0) means no timeout. The highest maximum value that can be entered is 10000 seconds. The default value is 600 seconds.
Number of Connections	Reject new connections after the specified number of connections are active. A value between 1 and 20 must be entered. The default value is 1 connection.
Security	

Parameter	Definition
Allowable Clients	If supplied, only messages from this list of fully qualified host names and/or IP addresses are accepted. Enter as a comma-separated list or use the _file() function.
Security Type	Select one of the following values from the drop-down list: ❏ **none.** Implies that the connection and command stream are not encrypted. ❏ **ssl.** Wraps the connection and command stream in an encrypted Secure Socket Layer (SSL). ❏ **ssh.** Provides secure shell (SSH) encryption and packet handling. The default value selected is *none*.
Client Authentication	If set to *true* and when the Security Type parameter is set to *ssl*, then the client's certificate must be trusted by the Telnet server for a connection to be created. Not used when the Security Type parameter is set to *none* or *ssh*.
Authentication Realm	When the Security Type parameter is set to *none* or *ssl*, the specify the name of a configured authentication realm to validate logins. For full access to management commands, the user must be assigned the *admin* role. If not supplied, logins will be delegated to the web console's user database. Not used when the Security Type parameter is set to *ssh*. For SSH console, authentication options are configured in the SSH provider.
Security Provider	Required if security is enabled (Security Type parameter value of *ssl* or *ssh*). This security provider will be used to secure the channel. When the Security Type parameter is set to *ssl*, then specify the name of an SSL Context Provider. When the Security Type parameter is set to *ssh*, then specify an SSH Provider.
Events	
Channel Failure Flow	Name of a published process flow to run if this channel cannot start or fails during message use. The server will attempt to call this process flow during channel close down due to the error.

Parameter	Definition
Channel Startup Flow	Name of a published process flow to run prior to starting the channel.
Channel Shutdown Flow	Name of a published process flow to run when the channel is shut down.

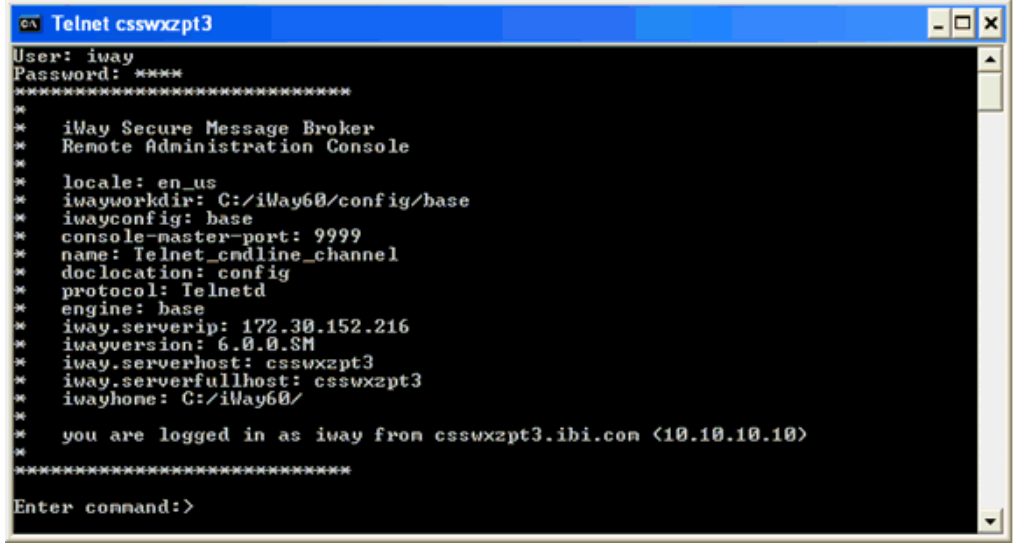
Connecting to a Remote Command Console

After you have configured a Telnet remote command console, you can use any command line Telnet client. Consider the following use case scenarios where you need to test iWay Functional Language (iFL) functions or browse help remotely for iWay Service Manager (iSM). The specific use of your Telnet client may vary, and users are referred to their specific Telnet client documentation. The Telnet client is not provided by iWay.

1. Connect to iSM using the command line. For example:

   ```
   telnet csswxzpt3
   ```

2. Enter a user name (for example, iway) and a password (for example, iway).

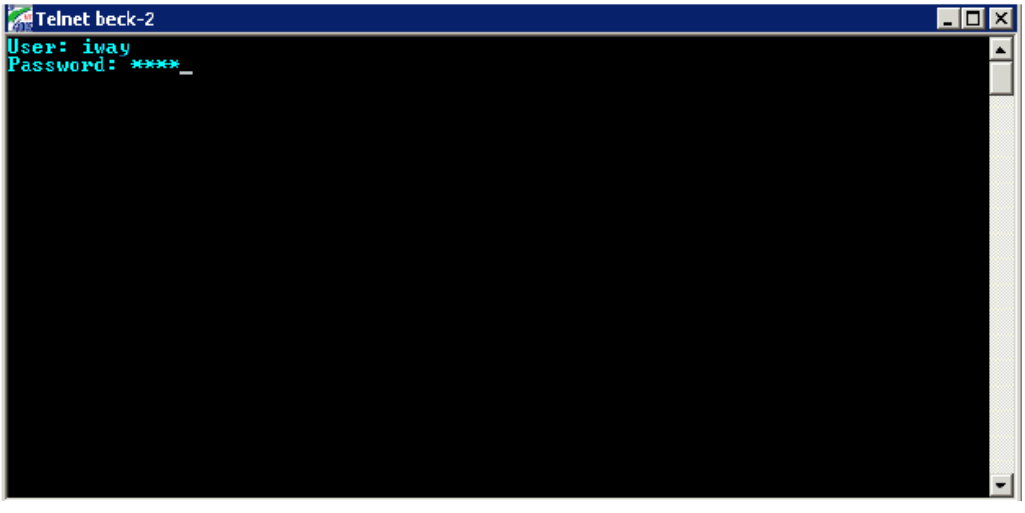


3. Once you are connected and logged in, you can now issue any command to monitor or control your iSM instance.

Using a Telnet Client

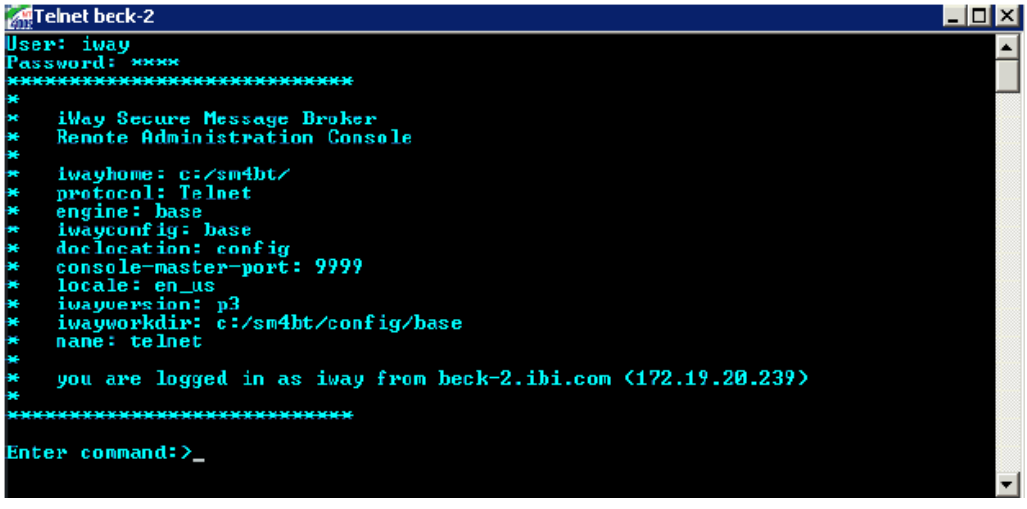
In this section, the default Telnet client that is available on Windows is used for demonstration purposes.

Once you start the Telnet client, the following Telnet logon screen is displayed, as shown in the following image.

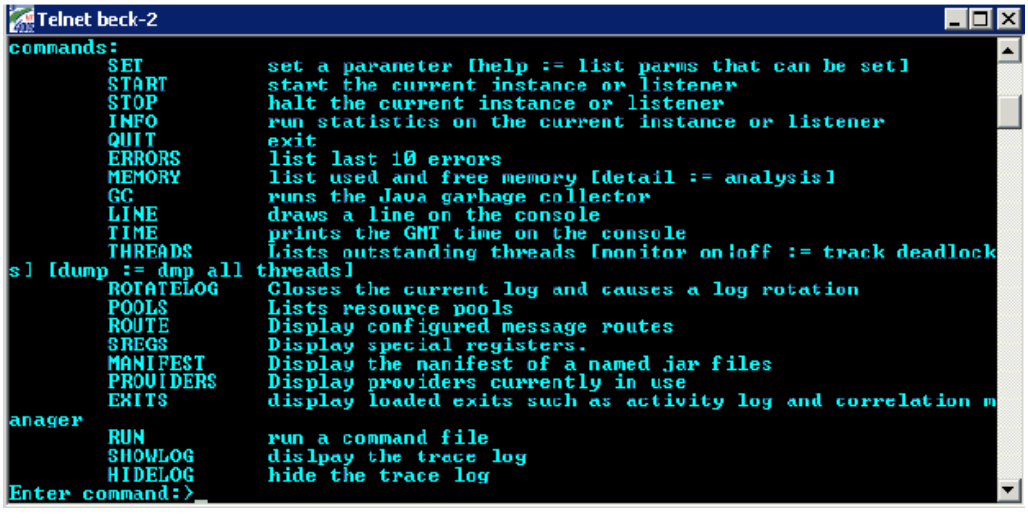


Provided that the connection meets the selected security criteria you are prompted for a user ID and password. These must be configured in the iSM Administration Console, and may have administrative capabilities or not. Lack of administrative capability means that commands that reconfigure iSM, such as *start*, *stop* and *reinit* are not available.

Once the logon is accepted, you are presented with a standard information screen, as shown in the following image.

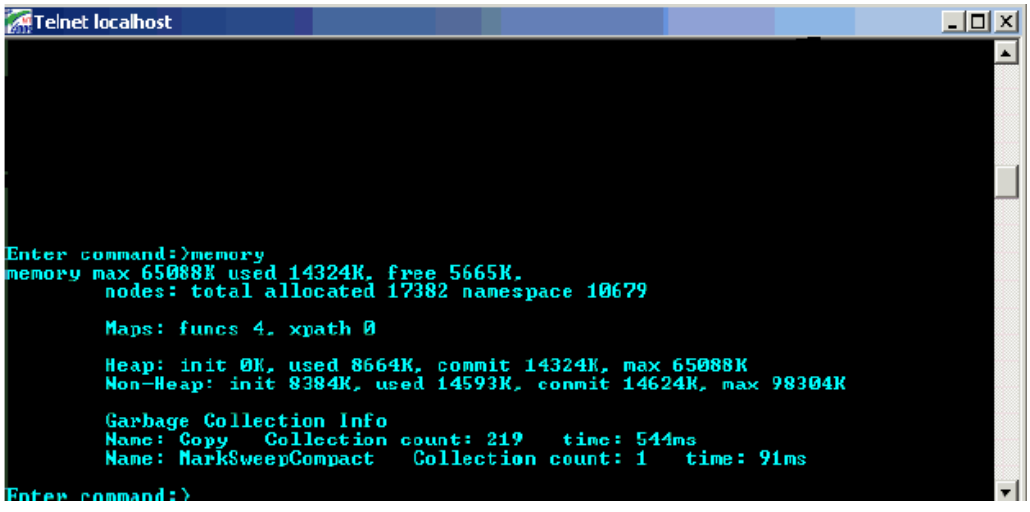


At the command line, you can use any authorized command. The *help* command lists these commands, as shown in the following image.



These are the same commands that can be issued from the standard shell console, plus the *showlog* and *hidelog* commands to enable or disable tracing for this Telnet session.

For example, if you enter the *memory* command, the following screen is displayed.



Remote Only Commands

The following iSM commands are available only from remote command consoles:

- **showlog.** Causes the trace log to be sent to the remote console.
- **hidelog.** Causes traces to not be sent to the remote console.

For more information on all of the commands that are supported for iSM, see the *iWay Service Manager Command Reference Guide*.

Telnet Scripting Example

The following is an example of automation or lights out operations that you can achieve after configuring a remote command facility using Telnet. A shell script is created containing the following command:

```
#!/bin/sh
host=localhost
port=9023
cmd="info"
( echo open ${host} ${port}
sleep 1
echo "iway"
sleep 1
echo "iway"
sleep 1
echo ${cmd}
sleep 1
echo quit ) | telnet > /home/jay/out.txt
echo " "
echo "* * * command output start * * *"
cat /home/jay/out.txt
echo "* * * command output end * * * *"
echo " "
```

There are more complex ways of running Telnet on Linux than I/O redirection. For example, the command `expect` is designed to work with interactive commands.

The following example shows more of the script that can be parameterized as an information-only command, which does not affect the behavior or configuration of the server.

```
* * * command output start * * *
telnet> Trying ::1...
Connected to localhost.
Escape character is '^]'.

User: iway
Password: ****
***************************
*
*   iWay Secure Message Broker
*   Remote Administration Console
*
*   protocol: Telnet
*   engine: base
*   iway.serverip: 127.0.1.1
*   locale: en_us
*   iwayversion: 7.0.3
*   iway.serverhost: UbuntuVM
*   iwayworkdir: /iway/prog/7.0.3.36971/config/base
*   iwayconfig: base
*   console-master-port: 9999
*   iway.pid: 3392
*   iway.serverfullhost: UbuntuVM
*   iwayhome: /iway/prog/7.0.3.36971/
*   name: Telnet1
*   doclocation: config
*
*   you are logged in as iway from localhost (0:0:0:0:0:0:0:1)
*
***************************
Enter command:>info
                              completed   failed   active   workers   free
SOAP1
   http          -- active --       0        0        0        3        3
   file          -- active --       0        0        0        3        3
Telnet1          -- active --       0        0        1        1        0
Enter command:>quit
goodbye!
* * * command output end * * *
*
```

Chapter 6

Using Event and Startup Process Flows

This section describes how iWay Service Manager (iSM) Event and Startup process flows can be used for troubleshooting and debugging purposes.

In this chapter:

Event Process Flows

Event process flows can be executed when specific (defined) events occur in iWay Service Manager (iSM) or during message processing. The process flows must be published to the configuration (iWay Integration Application) and must be available for execution at the time that they are called.

The Event process flows can run under the following constraints:

- Communicate with the caller by passing a return code as the name of the End node. This is the same rule as is required for subflows of a regular process.
- Can only return a single document, which may or may not be meaningful to the caller.
- Cannot use Emit nodes, although Emit services are permitted. Emit nodes schedule emits for execution at a later time (asynchronous to the process flow), while Emit services emit directly when they are called.

Other restrictions may apply for individual Event process flows. All Event process flows are conditional, and must be configured for execution if their use is required.

The following Event process flows are described in this section:

- Server Startup
- iWay Business Activity Monitor (BAM) Database Loss of Access
- Channel Startup Failure
- Retry Expired
- Failed ReplyTo

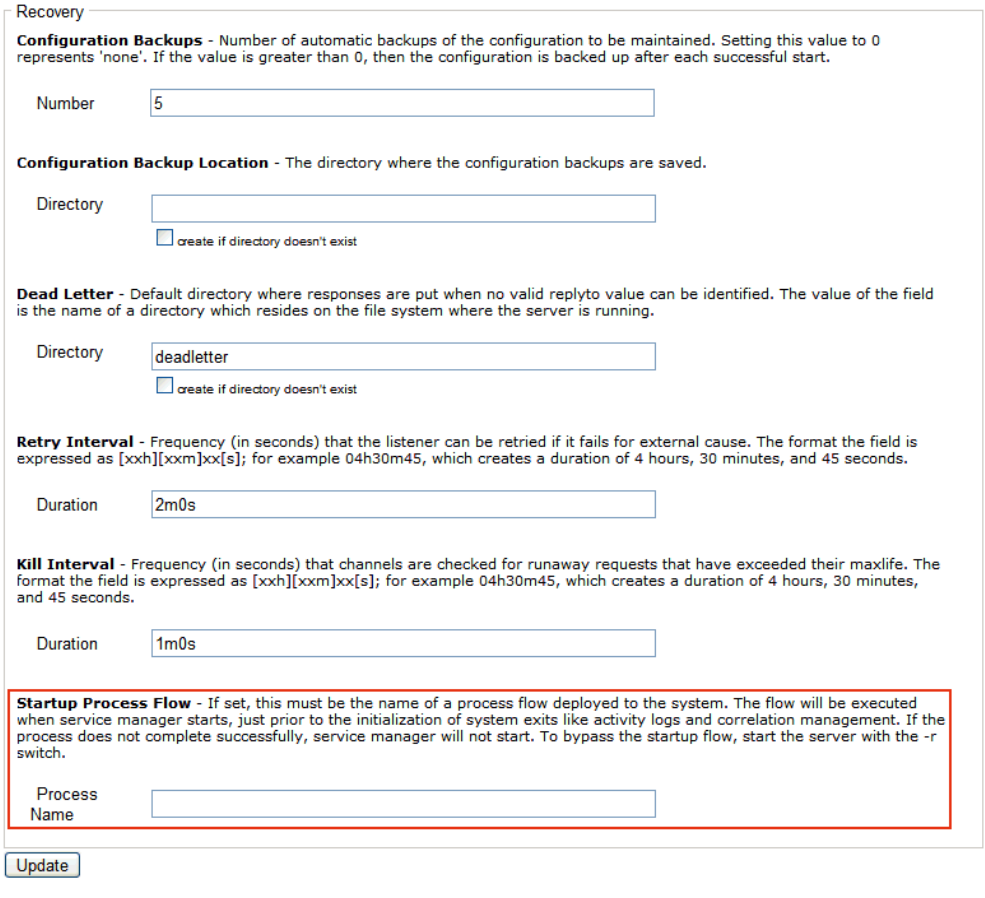
- ❑ Send to Dead Letter

Server Startup

The Server Startup process flow is executed by the iSM initialization routines as iSM starts its execution. This process flow can check for the availability of resources that are required by iSM, and can prevent iSM from starting if the resources are not available. A return of *success* allows iSM to continue its startup sequence. Otherwise the iSM startup is terminated.

The Server Startup process flow cannot start channels, since iSM is not ready to run channels at this early (startup) stage.

The name of the Server Startup process flow must be entered in the Recovery area of the General Settings page (Process Name field), as shown in the following image.



The following table lists and describes the possible edges that are returned by the Server Startup process flow.

Edge	Description
success	Continue with iSM startup.
<other> or flow fails	Do not continue to start iSM.

iWay Business Activity Monitor Database Loss of Access

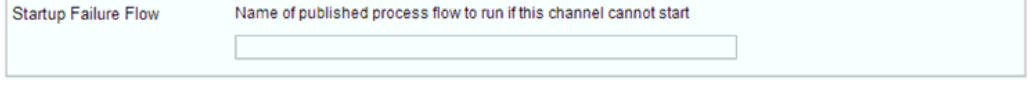
This process flow is executed when the iWay Business Activity Monitor (BAM) drivers lose connectivity to the BAM database. The process flow can notify an operation area of the problem, and can determine how iSM should continue:

- iSM continues, but BAM update is ignored.
- iSM terminates.
- iSM maintains a local file on disk containing BAM information and attempts to update the database when connectivity is restored.

Channel Startup Failure

The Channel Startup Failure process flow applies only to channels that are not started by a specific manual command. This process flow must be published to the system, and is executed whenever the channel cannot initialize. The process flow can be used to send an email to alert an administrator of the issue.

Enter the name of a published process flow to be executed in the Startup Failure Flow field, as shown in the following image. This field is a common channel property that is available for all iSM listeners.



The Channel Startup Failure process flow receives a signal message document for processing. The signal message document uses the following structure and format:

```
<channelfail  name='channelname' protocol='protocol' state='statecode'
statename='name of state'  failures='count' version='ism version'
time='timestamp' >
       <message>text of message</message>
</channelfail>
```

where:

`name`

Is the name of the configured channel.

`state`

Is a specific code describing the current state of the channel. The codes have assigned names, which are available in the statename attribute.

`statename`

Is the name of the current state, which will usually be one of the following:

- **config.** Cannot start due to a configuration error. The channel is not retried.
- **restart.** iSM will attempt restart.
- **stopped.** iSM will not attempt restart.

`protocol`

Is the name of the protocol being used by the channel (for example, File).

`failures`

Is the count of sequential failures (for example, base 1).

`version`

Is the version of iSM.

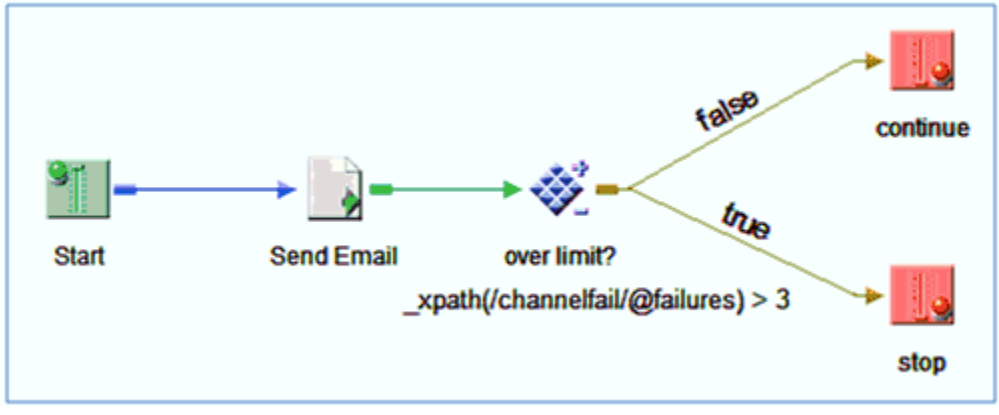
`time`

Provides a timestamp of the failure occurrence.

This process flow can signal iSM to stop retrying the channel by sending a *stop* message. This is done by naming the End node of the process flow (*stop*). Termination of the process flow by any other End node will instruct iSM to continue retrying the channel using the standard automatic retry logic.

The information in the signal message document passes information into the process flow concerning the channel and the most likely cause of failure.

In the following simplified example, a failure results in an email being sent to an identified party followed by a check to see if the number of sequential failures exceeds a designated limit (in this example, 3).



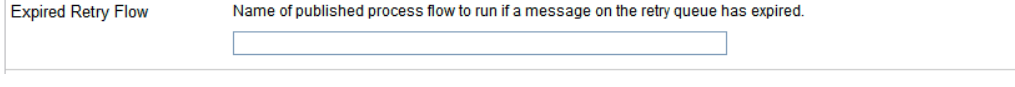
Normally this process flow would run during iSM startup or channel restart. To have the process flow run if the start is attempted from an iSM *start* command whether standalone or in a script, use the *-doflow* switch on the start command. For more information on using the start command, see the *iWay Service Manager User's Guide*.

Retry Expired

Messages can be queued for retry on channels that support this facility. This includes queue-based channels, the File channel, and the Internal Queue channel. The retries are triggered by logic in the process flow. In this circumstance, the message is re-executed on a periodic basis until expiration has been reached.

At the expiration point, a process flow can be executed to take recovery actions including notification, and optionally, changing the destination address or restarting with a changed (extended) expiration time.

Enter the name of a published process flow to be executed in the Expired Retry Flow field, as shown in the following image. This field is a common channel property that is available for all iSM listeners.



On entry, the process flow receives the document as it exists, at the point at which the process flow is called. The following table lists and describes several special registers that are available in the Retry Expired process flow to assist during the analysis.

Register Name	Description
iway.eventflow.exitflow	Identifies the purpose of the process flow (for example, *expiredRetry*).
iway.eventflow.attempts	Count of the number of retry attempts made before the expiration.
iway.eventflow.expiredtime	Time of the expiration.

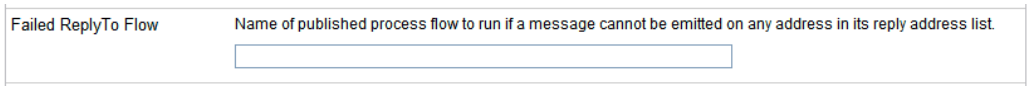
The following table lists and describes the possible edges that are returned by the Retry Expired process flow.

Edge	Description
success	The process flow overrules the expiration. iSM will attempt to resend, this time with the output of the process flow.
<other> or flow fails	An error document is sent to the error addresses.

Failed ReplyTo

A reply designation associated with a document triggers an emit operation following completion of the process flow. If the emit operation is not successful, the Failed ReplyTo process flow is triggered.

Enter the name of a published process flow to be executed in the Failed ReplyTo Flow field, as shown in the following image. This field is a common channel property that is available for all iSM listeners.



On entry, the process flow receives the document as it exists, at the point at which the process flow is called. The following table lists and describes several special registers that are available in the Failed ReplyTo process flow to assist during the analysis.

Register Name	Description
iway.eventflow.exitflow	Identifies the purpose of the process flow (for example, *failedReply*).
iway.eventflow.replyname	Configured name of the reply or error specification.
iway.eventflow.destination	The address configured for the emit, as evaluated for use.
iway.eventflow.errormsg	An error message, if any, describing the cause of the failure that caused this event to be generated.
iway.eventflow.replyprotocol	Protocol used for the emit attempt (for example, File, MQ, and so on).

The following table lists and describes the possible edges that are returned by the Failed ReplyTo process flow.

Edge	Description
success	The process flow took responsibility to deliver the message.
<other> or flow fails	An error document is sent to the error addresses.

Each ReplyTo and ErrorTo is treated separately. If an error occurs for one, an attempt is made to handle the error, and iSM continues with the rest of the list. Error handling, however, differs for ReplyTo versus ErrorTo.

A failed ReplyTo causes the Failed ReplyTo process flow to execute (if present). If the process flow is successful (by terminating at an End node called success), the error is considered to be handled and iSM continues through the rest of the address list. If the process flow is absent, fails, or reaches an End node with a different name, then iSM creates an error document and attempts to send it to the ErrorTo instances recursively. All ErrorTo instances will be called for each ReplyTo that fails.

Document siblings are treated as independent documents. The net effect should be similar to sending the document first, and then each of its siblings one by one. iSM does not expect error documents to contain siblings. However, if present, they too will be sent as top-level documents (which may or may not be in error).

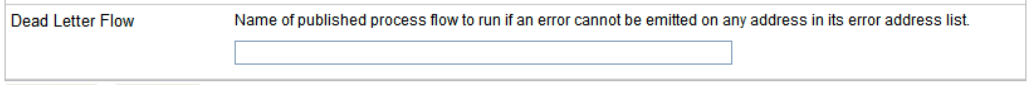
Send to Dead Letter

Messages queued for emitting at a later time (using the channel configuration (called ReplyTo and ErrorTo) or the Emit object in a process flow are sent when the outlet of the channel is executed. Messages can also have alternate addresses if required.

If all attempts to emit the message fail, then by default, the message is written to a configured *dead letter* directory.

If an *emit failed* process flow is configured, then the process flow can examine the message, redirect it, replace it, and potentially notify an appropriate authority. It can then send the message to another channel for a retry attempt or continue to allow the message to be written to the dead letter queue.

Enter the name of a published process flow to be executed in the Dead Letter Flow field, as shown in the following image. This field is a common channel property that is available for all iSM listeners.



On entry, the Send to Dead Letter process flow receives the document as it exists at the point at which the process flow is called. The following table lists and describes several Special Registers (SREGs) that are available in the process flow to assist during the analysis.

Register Name	Description
iway.eventflow.exitflow	Identifies the purpose of the process flow (for example, *deadLetter*).
iway.eventflow.replyname	Configured name of the reply or error specification.
iway.eventflow.destination	The address configured for the emit, as evaluated for use.
iway.eventflow.errormsg	An error message, if any, describing the cause of the failure that caused this event to be raised.

Register Name	Description
iway.eventflow.replyprotocol	Protocol used for the emit attempt (for example, File, MQ, and so on).

The following table lists and describes the possible edges that are returned by the Send to Dead Letter process flow.

Edge	Description
success	The message was successfully handled.
<other> or flow fails	The output of the process flow to be written to the dead letter directory, if configured.

Each ReplyTo and ErrorTo is treated separately. If an error occurs for one, an attempt is made to handle the error, and iSM continues with the rest of the list. Error handling, however, differs for ReplyTo versus ErrorTo.

A failed ReplyTo causes the Failed ReplyTo process flow to execute (if present). If the process flow is successful (by terminating at an End node called success), the error is considered to be handled and iSM continues through the rest of the address list. If the process flow is absent, fails, or reaches an End node with a different name, then iSM creates an error document and attempts to send it to the ErrorTo instances recursively. All ErrorTo instances will be called for each ReplyTo that fails. ErrorTo instances are used to communicate errors to administrators who are able to resolve such situations.

A failed ErrorTo causes the Send to Dead Letter process flow to execute (if present). If the process flow returns success, iSM considers the error to be handled and continues with the rest of the address list. If the process flow is absent, fails, or reaches an End node with a different name, then iSM attempts to write a file under the configured dead letter directory.

Sending an error to an empty list of ErrorTo instances is an error. It is handled the same way as a failed ErrorTo.

Notice that only error documents are sent to the configured dead letter directory. If an error cannot be reported (because an ErrorTo fails or there are no ErrorTo instances), then iSM attempts to send the error document to the dead letter directory to keep a record for manual processing. An error document contains a copy of the original document that generated the error.

iSM attempts to avoid sending to a duplicate address within the list if iSM already knows it is a bad address. This could happen when an ErrorTo is also a ReplyTo. A duplicate bad address is treated the same as a regular failed ReplyTo or ErrorTo, except the IO was never attempted.

Document siblings are treated as independent documents. The net effect should be similar to sending the document first, and then each of its siblings one by one. iSM does not expect error documents to contain siblings. However, if present, they too will be sent as top-level documents (which may or may not be in error).

Parse Failure

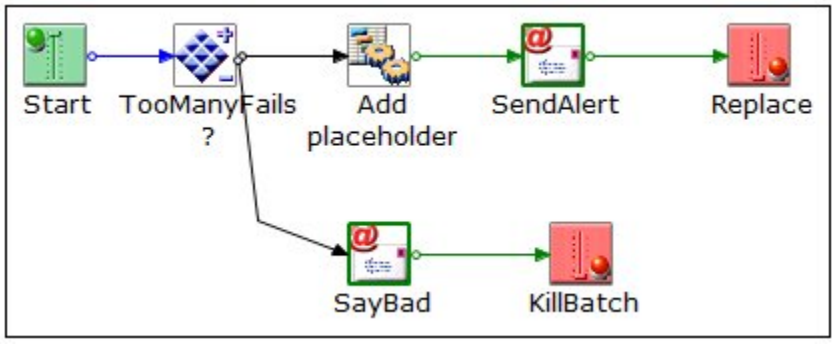
The Parse Failure flow is invoked if an incoming message fails the *parse to XML* operation for a channel. This does not apply to a parse that is handled within a process flow by a service (agent) for that purpose.

The incoming document to the flow contains the message that failed parsing. The standard Special Registers (SREGs) for the protocol are available in the flow. For example, a bad message on a File listener will provide the usual information on the source of the file.

The Parse Failure flow can also be used to send a notification.

The flow can replace the document that could not be parsed. This might be done to *fill in* an element in a large batch managed by a splitting preparser. To replace the message, set the document on output to the message required, and return through an End node named *Replace*. The replaced message will then pass through the normal channel cycle. It may be necessary in your application to set a SREG in order to notify subsequent processes that this is a *placeholder* message. If this technique is used, then remember to set the SREG at the channel level, so as to make it available beyond the scope of the flow.

On entry to the event flow, the SREG *iway.parsefail* will be set to the count of the number of parse failures in this channel for this transaction. This count is useful for batch handling, in which a splitting preparser divides the batch into a sequence of sub-messages. For example, your flow might determine that the count of *placeholder* messages returned to the channel has exceeded a threshold, and so elects to take application action to reject the batch.



Startup Process Flow

The Startup Process Flow optionally executes as iSM starts. The name of the process flow is entered in the *Recovery* area of the console. If named and present, the process flow is executed by the server just prior to the installation of system components. For example, if SNMP did not begin, then the process flow itself will not be recorded in the activity logs.

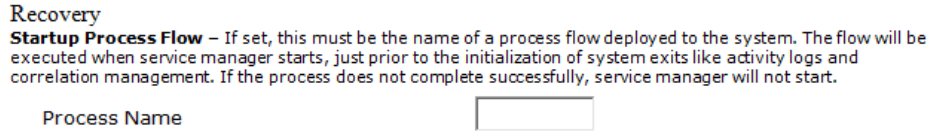
If the process flow ends successfully, the server continues with its startup process. If the process does not end successfully (for example, a fail service is encountered), the server does not start.

The process flow is designed to enable the server to verify the availability of required resources. For example, an SQL service in the process flow may perform a simple select against the Business Activity Monitor (BAM) tables by accessing the jdbc/BAMDBProvider. If the select fails, it can be assumed that the BAM database is not available, and the process flow issues a fail. This would prevent processing if BAM, deemed by the application designer to be a critical resource, is not available. Similarly, if an application required the transfer of data from an Oracle to a DB2 database, the startup process flow could determine that both are available before allowing the server to start. Startup criteria are at the discretion of the application designer.

Once started, the server manages errors and recovery normally.

You cannot control the server from this process flow. For example, you cannot use the control service to start channels because the server has not yet been sufficiently initialized for channels to properly start. Other facilities, including the autostart script, can be used for this purpose.

The following image shows the Recovery pane.



On entry, the input document to the process flow is shown below:

```
<startup version=currentversion time=timestamp/>
```

where:

currentversion

Is the server version number, such as 7.0.

timestamp

Is a standard RFC 3339 (ISO 8601) timestamp.

The output document is ignored.

The startup parameter -r causes iSM to start without calling the startup exit. This allows a *buggy* startup exit to be bypassed so that iWay tools can be used to correct any problems.

Note: This is available under the batch (manual) startup mode. Users are advised to avoid starting as a service until the startup exit is known to be functioning properly.

Chapter 7

Recommended Third-Party Tools for Troubleshooting and Debugging

This section provides information on iWay-recommended third-party tools if the troubleshooting or debugging level scope falls outside of the iWay framework.

In this chapter:

JConsole

The Java Monitoring and Management Console (JConsole) can be used to provide information on iWay Service Manager (iSM) performance and resource consumption running on a Java platform. The JConsole uses Java Management Extension (JMX) technology.

JConsole provides a visual (graphical) representation of the Java Virtual Machine (JVM) environment where iSM is running.

Specifically, JConsole provides time-range charts showing usage for the following JVM components:

- Memory Heap Usage
- Threads
- Classes
- CPU Usage

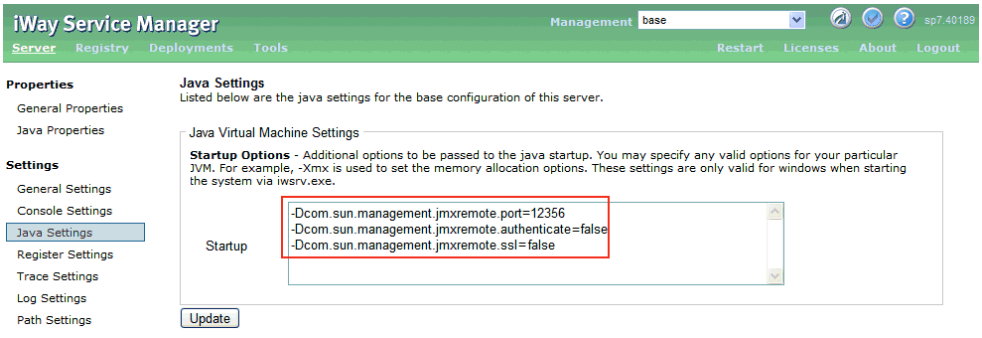
For more information on configuring and using JConsole, see:

http://docs.oracle.com/javase/6/docs/technotes/guides/management/jconsole.html

Procedure: How to Enable iWay Service Manager for Monitoring Through JConsole

1. Log in to the iWay Service Manager Administration Console.
2. Click *Java Settings* in the left pane.
3. Specify the following Java startup options:

```
-Dcom.sun.management.jmxremote.port=12356
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```



4. Click *Update*.

   Note: Use caution when modifying the Java settings. If you make a mistake, then iSM may not start.

5. Stop and start iSM.
6. Using JConsole, access iSM by entering the host name and port.

JVM Startup Options

The following JVM startup options can be used to control the Heap size.

Initial Heap Size:

```
-Xms256m
```

Maximum Heap Size:

```
-Xmx256m
```

The following JVM startup options can be used for JVM OOME debugging:

```
-XX:-HeapDumpOnOutOfMemoryError
-XX:-HeapDumpPath=./java_pid<pid>.hprof
```

SoapUI

SoapUI is a free, open source, and cross-platform functional testing solution. In a single testing environment, SoapUI provides complete test coverage and supports all of the standard protocols and technologies.

SoapUI can be used:

- During the development of web service clients that need to be tested on an interactive level.
- For baseline, load, and soak testing strategies.
- For fixed rate testing strategies.
- For variable load testing strategies.
- During statistics calculation and thread count changes.
- When simultaneously running multiple load tests.

For more information on configuring and using SoapUI, see:

http://www.soapui.org/

KeyTool IUI

KeyTool IUI is a cryptography GUI tool that allows you to configure keys and certificates, including the ability to verify, sign, encrypt, and decrypt the files.

KeyTool IUI provides you with the following functionality:

- Exporting and importing trusted certificates
- Creating DSA and RSA keypairs
- Creating a CA certificate reply to a RSA keypair
- Exporting CSR from a RSA keypair
- Creating empty keystore functions

KeyTool IUI displays detailed information in the GUI about private keys (keypairs) and trusted certificates, regarding their valid date, self-signed, trusted CA, key size, certificate type, certificate signature algorithm, or modified date.

For more information on configuring and using KeyTool IUI, see:

http://code.google.com/p/keytool-iui/

Operating System Commands

Another recommendation is to use operating system commands (for example, Netstat and Traceroute), which are also effective for troubleshooting and debugging purposes.

Netstat is a command-line tool that displays network connections (incoming and outgoing), routing tables, and a number of network interface (network interface controller or software-defined network interface) and network protocol statistics. For more information, see:

http://en.wikipedia.org/wiki/Netstat

Traceroute is a network diagnostic tool for displaying the route (path) and measuring transit delays of packets across an Internet Protocol (IP) network. For more information, see:

http://en.wikipedia.org/wiki/Traceroute

Wireshark

Wireshark is a free and cross-platform compatible packet analyzer. It is used for network troubleshooting and analysis. Wireshark is similar to *tcpdump*, but has a graphical user interface and some integrated sorting and filtering capabilities.

For more information on configuring and using Wireshark, see:

http://www.wireshark.org/

Tcpdump

A fairly common packet analyzer, tcpdump is also free and cross-platform compatible. Tcpdump analyzes network behavior, performance and applications that generate or receive network traffic. It can also be used for analyzing the network infrastructure itself by determining whether all necessary routing is occurring properly, allowing the user to further isolate the source of a problem.

For more information on configuring and using tcpdump, see:

http://www.tcpdump.org/

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

TIBCO, the TIBCO logo, the TIBCO O logo, FOCUS, iWay, Omni-Gen, Omni-HealthData, and WebFOCUS are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.