TIBCO iWay® Service Manager

Cross-Channel Services Guide

Version 8.0 and Higher
March 2021
DN3502287.0321

Contents

Chapter 1

Introducing iWay Service Manager Cross-Channel Services

This section provides an introduction to iWay Service Manager (iSM) cross-channel services. For more information on additional queuing protocol adapters that are supported by iSM, see the *iWay Service Manager Protocol Guide*.

In this chapter:

Cross-Channel Services Overview

iWay Service Manager (iSM) provides channels that link processes within iSM to other processes in the same or another instance of iSM. This contrasts with channels that acquire messages from external media, such as a File or FTP connection. These channels divide the message execution process into stages for application purposes, such as:

- Exchanging messages with other servers.
- Dividing the application into more manageable portions.
- Changing threading models based on specific sections of the application.
- Performance tuning.

Cross-channel message exchanges can be used to facilitate application modularization. In addition to simplifying application development and maintenance, an application that is composed of modular sections provides horizontal and vertical scaling. The modular sections can be executed:

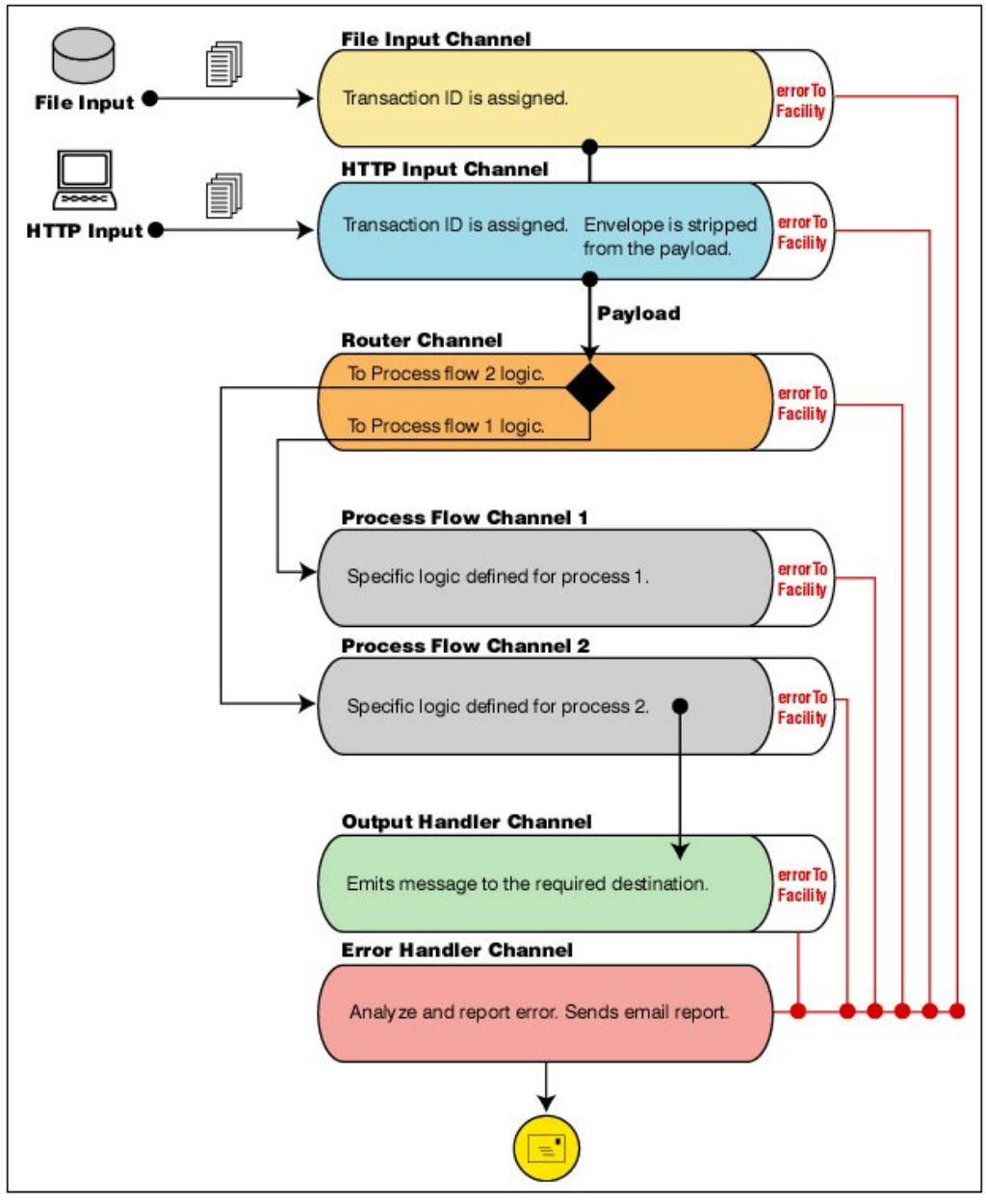
- Within the same JVM, which facilitates the selection of the appropriate threading model for that portion of the application.
- Across JVMs on the same computer to take advantage of the dispatching mechanisms that belong to this computer.
- Across computers, which allows several computer systems to work on the application simultaneously.

The mechanisms that support cross-channel message exchanges provide opportunities for workload balancing and backup/recovery when used in appropriately designed applications.

The application design that is best oriented for the use of cross-channel distributions is called the multi-channel architecture, which is a common architecture for iSM-based applications. For example, consider the following modularized breakdown of a typical application:

1. A message arrives from an external source. This message is examined and transformed, routed to a section of the application that processes the payload, and then passed to a section that emits the final result.
2. A special error handler is configured to report on application issues that are not handled specifically in other portions of the application.

3. An error handler channel receives messages through the *errorTo* facility of each channel using an Internal emit service to direct the error messages that are not handled within the application channel.



The following channel types are available:

- **Internal.** Passes messages between channels for asynchronous or synchronous execution. For more information, see *Internal Queue Processing* on page 11.
- **Ordered.** Passes messages between channels for asynchronous execution, maintaining execution order and batch control. For more information, see *Ordered Queue Processing* on page 27.
- **Reverse Invocation (RVI).** Exchanges messages between two or more instances of iSM, with support for reverse connections. This allows full protection of the iSM instances behind outbound-only firewalls. For more information, see *Reverse Invocation Queue Processing* on page 51.
- **Asynchronous Forward Transfer Invocation (AFTI).** Marshalls and unmarshalls messages and their context to be exchanged over protocols other than the existing cross-channel protocols that are provided by iSM. For more information, see *Asynchronous Forward Transfer Invocation Queue Processing* on page 75.

All of the cross-channel links pass messages and full execution context. Additionally, all messages and contexts can be compressed and encrypted for secure processing.

Available Listeners Reference

The following table provides a quick reference to the iSM listeners that are defined in this documentation for cross-channel services.

Listener Name
Internal Queue (See *Configuring an Internal Queue Listener* on page 12.)
Ordered Queue (See *Configuring an Ordered Queue Listener* on page 28.)
RVIAttach (See *Configuring the RVIAttach Listener* on page 61.)
RVIGateway (See *Configuring the RVIGateway Listener* on page 64.)

Available Services Reference

The following table provides a quick reference to the iSM services that are defined in this documentation for cross-channel services.

Service Name
Internal Emit Agent (com.ibi.agents.XDInternalEmitAgent) (See *Configuring an Internal Emit Service* on page 17.)
Ordered Emit Agent (com.ibi.agents.XDOrderedEmitAgent) (See *Configuring an Ordered Emit Service* on page 39.)
Marshalls a message Agent (com.ibi.agents.XDMarshallAgent) (See *Configuring a Marshalls a Message Service* on page 77.)
RVI Relay (com.ibi.agents.RVIRelay) (See *Configuring the RVI Relay Service* on page 62.)
Unmarshalls a message Agent (com.ibi.agents.XDUnmarshallAgent) (See *Configuring an Unmarshalls a Message Service* on page 78.)

Chapter 2

Internal Queue Processing

This section describes how to configure Internal queue processing.

In this chapter:

Internal Queue Processing Overview

Some designs require that a message be passed from one process to another. For example, a process might receive a message on a protocol, process it, and then desire another listener (thread) to complete the message operation. Frequently, such messages are stored in the file system using the File emit service, and then picked up using the File listener.

As an alternative, iWay Service Manager offers the Internal Queue listener. Messages are stored in memory by the Internal emit service and held until picked up by the Internal Queue listener. Messages can be made to persist in the event that the server fails or is terminated before all held messages are processed.

Each listener is associated with one internal queue that has been assigned a name. The flow that is required to store a message in the queue configures an Internal emit service to that assigned name. The listener picks up the stored messages by First In, First Out (FIFO) method, and passes them for execution.

As the queue grows and shrinks, the Internal Queue listener can manage the listeners feeding the queue to maintain an average, desirable size. This is referred to as rubber banding or back pressure. The low and high mark configuration values control this.

A common use of the internal emitter is to pass a message to one or more work flows that operate asynchronously. For example, each Internal Queue listener flow may update a different database or may access records from a single database based upon a computed modulus key.

As with all protocols, it is possible to either emit to the protocol following completion of the message flow or to emit immediately using an emit service (com.ibi.agents.XDInternalEmitAgent) within the flow, which can be configured using a Queue object for Internal Emit. If the service approach is used, then the handling of the message by the Internal Queue listener can proceed asynchronously with the flow that originated the message.

Once a message has been passed to an Internal Queue listener, it is separated from its originator and the originator can neither await its execution or obtain response values.

The following is a list of additional scenarios where Internal Queue processing can be implemented:

- Dividing an application into multiple steps simplifies development efforts. iWay Business Activity Monitor (BAM) can restart the message processing between steps.
- There are several applications that must emit to SAP through a single access connection.
- HTTP messages are currently being received on many slow speed lines. There is a requirement to multiplex the messages for effective use on a single high speed line to a third party.
- A high performance application needs to take advantage of channel throttling and control between application steps to improve overall throughput.
- There is a requirement to consolidate error handling into a single channel.

Configuring an Internal Queue Listener

To configure an Internal Queue listener, you must create a channel for your application project using the Channel Builder in iWay Integration Tools (iIT) and select *Internal Queue* as the listener type for your inlet. For more information, see *Configuring a Listener Using iWay Integration Tools* on page 81.

For a complete description of the configuration parameters that are available for the Internal Queue listener, see *Internal Queue Listener Configuration Parameters* on page 12.

For a complete description of the Internal Queue listener Special Registers (SREGs), see *Internal Queue Listener Special Registers* on page 16.

Reference: Internal Queue Listener Configuration Parameters

The following table lists and describes parameters for the Internal Queue listener.

Parameters that are common to queue listeners are described in *Listener Configuration Parameters* on page 95.

Property	Description
Name of Internal Queue (required)	A simple, case-sensitive name used to tie the emit service and the listener. This name must be unique to the listener, but can be specified as the destination for any number of emit services.
Persistent	If enabled, messages are persisted. Persistent messages are held in the safestore until completion and can be recovered if the server is restarted. Select one of the following options from the drop-down list: ❏ none {false} ❏ rdbms {rdbms} ❏ file {true} The default value is none {false}. The *rdbms* option enables the application to persist messages to a remote database location where, in the event of a failover situation, a secondary server running remotely can continue processing persisted messages without any business interruption. To use RDBMS persistence, you must create the iway_queues table in the JDBC provider, using the DDL script in the etc/setup directory. A hot backup channel can be configured to use the same RDBMS safestore.
Safestore Location	If persistent, this is the location in the file system to which documents are safestored.

Property	Description
Compress Persistent	If the queue is persistent, the documents written to the safestore can be compressed. Select one of the following options from the drop-down list: ❑ smallest {best} ❑ none {none} ❑ fastest {speed} ❑ standard {std} The default value is to not compress (none).
Low Mark	If the size of the named queue falls below this value from above the value, the named listeners in the Control List are sent an activate message. The default value is 0.
High Mark	If the size of the named queue goes above this value, a passivated message is sent to the listeners in the Control List. The default value is 0.
Control List	If a high or low mark value is crossed, the appropriate message is sent to each listed listener. For example, if two listeners LISA and LISB are feeding the internal queue, listing them as LISA,LISB will cause each to receive the appropriate message.
Inhibit Add	If set, the queue will not accept new messages when its size reaches the high mark, and will resume accepting messages when the number of messages on the queue reaches the low mark. The effect of this inhibition can cascade through the application, controlling overall performance. The default value is false.

Property	Description
External Mark	When set to a number greater than 0, this causes the message to be stored in memory up to the specified number. When the queue grows larger than the specified number, the message is stored on disk. This is equivalent to splitting the memory and performance optimization for the messages in the queue, and is effective for large lists that may fill memory. The default value is 0.
Duration	Maximum time in seconds (allows xhxxmxxs format) that a document can remain in this channel, starting from when the document is added to the queue, including pends and retries. The default value is 24 hours.
Support Pending	If set, messages can be pended for later execution if the process flow calls for a Fail/Pending operation. Pending messages persist for a specified time and are retried at a specific interval. You might use a pending operation in the event that a message cannot be processed because an external resource is currently unavailable. The default value is false.
Retry Interval	Determines the interval (in seconds) between retrying pending requests. The default value is 600 seconds (10 minutes).

Note: Passivation and Inhibition affect the threading model and the movement of messages within an application. Using these facilities, performance can be improved by avoiding queue congestion. For more information on passivation and inhibition, see Chapter 1, *Introducing iWay Service Manager* in the *iWay Service Manager User's Guide*.

Reference: Internal Queue Listener Special Registers

The following table lists and describes the Special Registers (SREGs) available on the Internal Queue listener. These values can be used in the application for message routing and processing.

Name	Source	Level	Type	Description
iway.channel	Listener	System	String	Full name of the channel (may include channelname.inlet.listener).
iway.channelname	Listener	System	String	Channel name portion of the name from the full channel name of channelname.inlet.listener.
iway.inletname	Listener	System	String	Inlet name portion of the name from the full channel name of channelname.inletname.listener.
iway.listener	Listener	System	String	Name of the listener.
iway.pid	System	System	String	Process ID of the server, if available.
iway.serverfullhost	System	System	String	Full host name of the server (includes domain).
iway.serverhost	System	System	String	Host name of the server.
iwayconfig	System	System	String	Current active configuration name.
iwayhome	System	System	String	Base at which the server is loaded.
iwayversion	System	System	String	Release version of the server.
iwayworkdir	System	System	String	Path to the base of the current configuration.

Name	Source	Level	Type	Description
msgsize	Listener	Document	Integer	Physical length of the message payload.
name	Listener	System	String	Assigned name of the master (same as iway.channel).
protocol	Listener	System	String	Protocol on which the message was received.

Configuring an Internal Emit Service

Messages are sent to particular destinations at the completion of a workflow. The state of the document determines which particular destination is used. The order in which the destinations are used cannot be predicted.

To route an output document or error message to a protocol other than that of the listener destination, you must configure an emit service in your application or an emitter as part of the Channel Outlet. For example, an application can receive input over FTP, but want to route the output to an Internal Queue listener.

The associated emit service is used to place messages onto the internal queue. Enter the name of the queue as the destination. The queue you specify must be an Internal Queue listener that has already been defined.

It is also possible to emit to multiple queues in a single emit operation. To do this, list the queues, separated by commas (,). For example, internalqueue1, internalqueue2, internalqueue3. This will emit to three queues, which are handled by three Internal listeners.

For a complete description of the configuration parameters that are available for the Internal emit service, see *Internal Emit Service Parameters* on page 23.

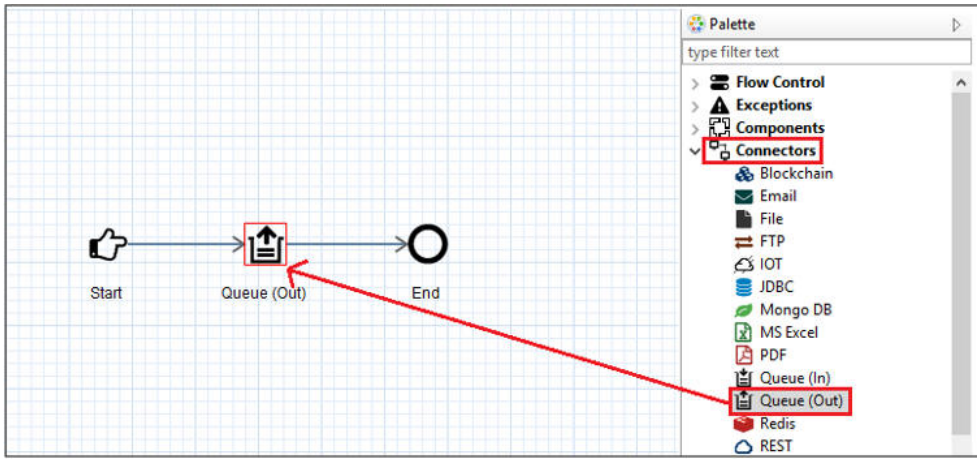
For a complete description of the edges that are returned by the Internal emit service, see *Internal Emit Service Edges* on page 24.

Procedure: How to Configure an Internal Emit Service

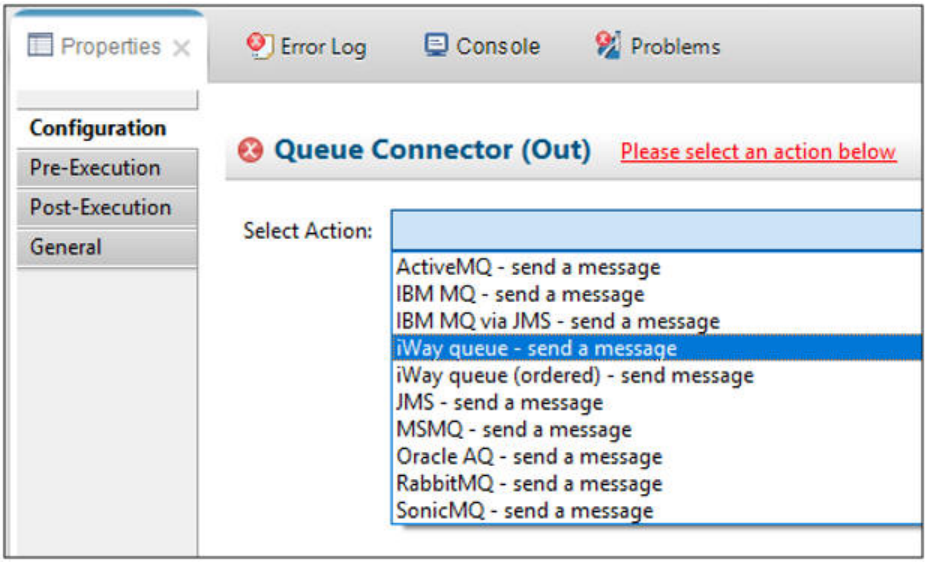
To configure an Internal emit service, you must create a process flow in your application project using iWay Integration Tools (iIT) and use the Queue (Out) object from the Palette, under Connectors, to emit to the internal queue (*iWay queue - send a message*) action.

Note: The Queue Out object implements the *iWay queue - send a message* action using the Internal emit service (com.ibi.agents.XDInternalEmitAgent).

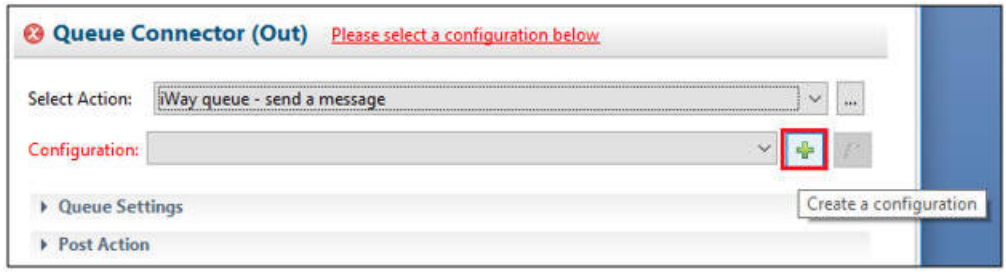
1. Expand the *Connectors* category in the Palette and drag the *Queue (Out)* object to your process flow, as shown in the following image.



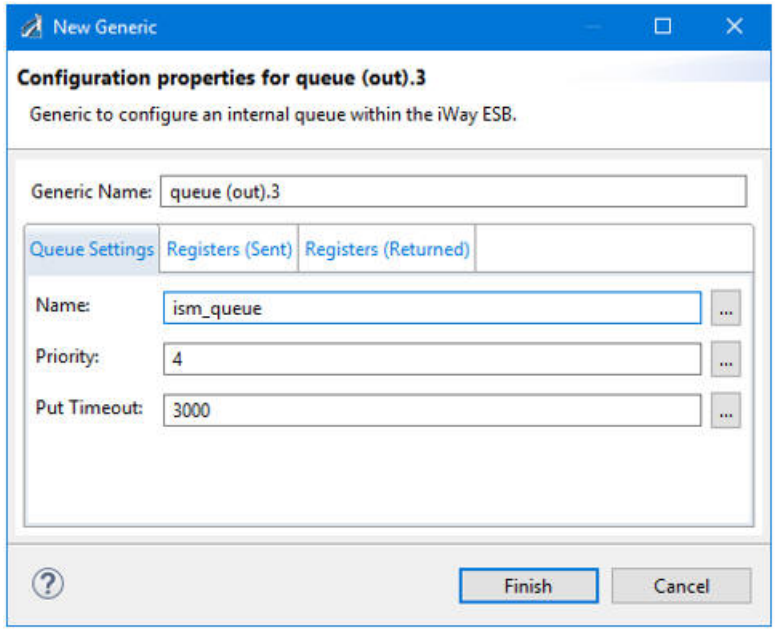
2. In the Properties tab, under Configuration, select *iWay queue - send a message* from the Select Action drop-down list, as shown in the following image.



3. Click the *Create a configuration* icon to the right of the Configuration field to configure a new generic for this object, as shown in the following image.



The New Generic dialog box opens, as shown in the following image.

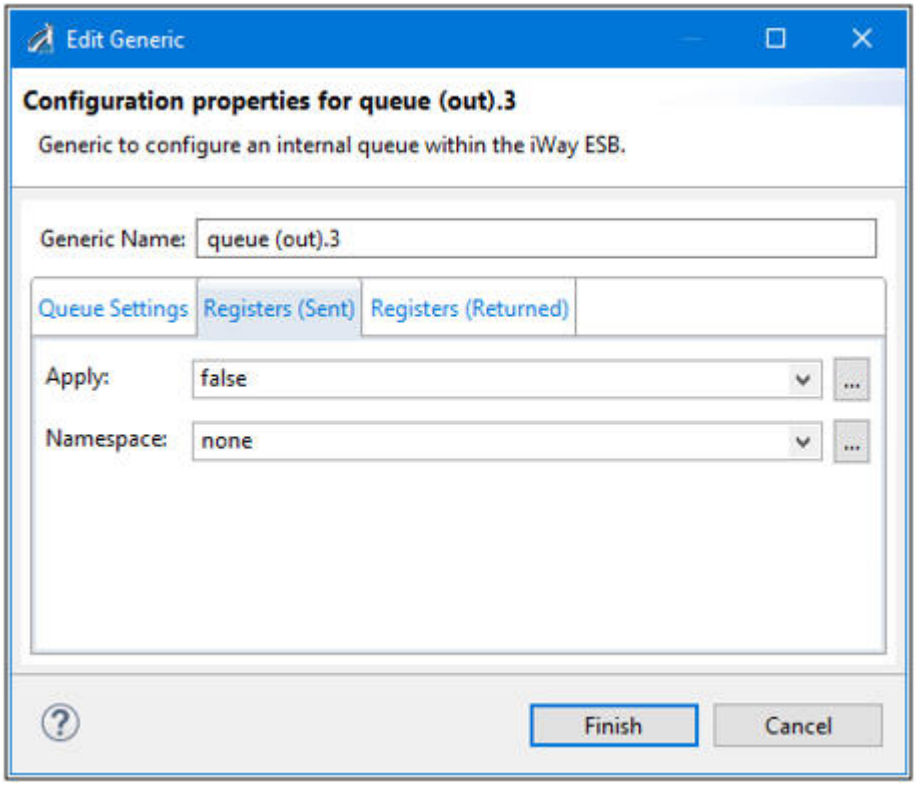


4. Specify a name for this generic in the Generic Name field, or accept the default.
5. In the Queue Settings tab, specify values for the following configuration parameters:

   - **Name.** Name of the internal queue for which to post messages.
   - **Priority.** Priority is an integer between 0 and 9, inclusive. The lowest priority is 0. The highest, most expedited priority, is 9. The default value is 4.

- **Put Timeout.** Time, in milliseconds, to wait for the queue to become available when attempting to put a message on the queue. You can enter zero (0) for an unlimited wait, but this is not recommended. If no value is supplied, timeout will be set to 3000 milliseconds.

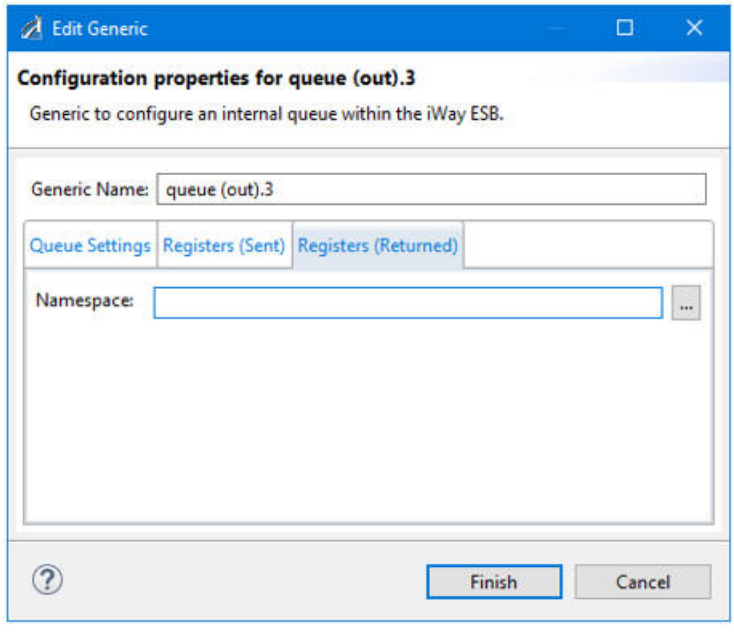
The Registers (Sent) tab is shown in the following image.



By default, the Apply parameter is set to *false*. User registers are processing variables and their values. If you want these registers to be emitted with the message, set this to *true*. In resubmit operations, this is set to *false*.

The Namespace parameter in this tab is used for synchronous or asynchronous processing. A namespace or list of namespaces containing registers will be made available to the Internal Queue listener. Select *Default Namespace* from the drop-down list for all registers in default (no prefix), *None* to send no registers, or *All* for registers from all namespaces. The default value is *none*.

The Registers (Returned) tab is shown in the following image.

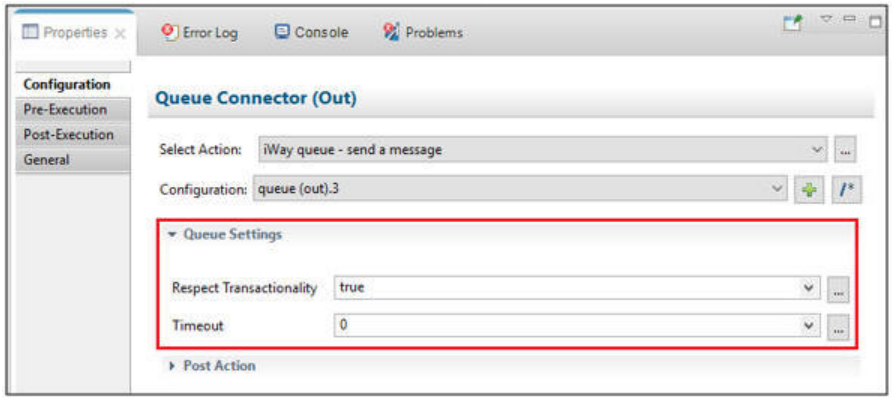


In synchronous processing, registers set by the Internal Queue listener in the return namespace will be made available to the calling process. If a value is supplied for the Namespace parameter in this tab, then response registers will be copied to the specified namespace. Leave this parameter blank to store response registers in the return namespace.

6. Click *Finish*.

   You are returned to the Properties tab.

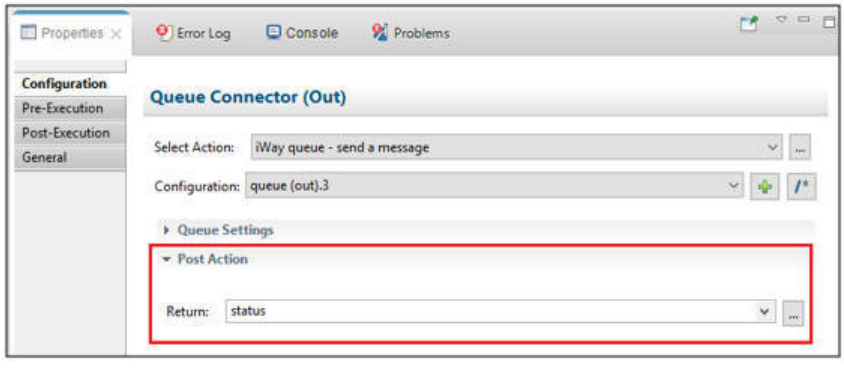
7. Expand *Queue Settings*, as shown in the following image.



8. Specify values for the following configuration parameters:

   - **Respect Transactionality.** Determines whether to respect existing transactionality. The default value is true.
   - **Timeout.** Determines how many seconds to wait for synchronous response. Set to zero (0) or leave blank to wait indefinitely. The default value is 0.

9. Expand *Post Action*, as shown in the following image.



10. Specify a value for the following configuration parameter:

    - **Return.** Select one of the following options from the drop-down list:
      - **status.** The status document will be the output document. This is the default value.
      - **input.** The input document will become the output document.

11. Save your process flow.

Reference: **Internal Emit Service Parameters**

The following table lists and describes parameters for the Internal emit service.

Note: Parameters that are common to emit services are described in *Service Configuration Parameters* on page 99.

Parameter	Description
Queue Name (required)	Name of the internal queue for which to post messages.
Want User Registers	User registers are processing variables and their values. If you want these registers to be emitted with the message, set this to true. In resubmit operations, this is set to false. The default value is false.
Priority	Priority is an integer between 0 and 9 inclusive. The lower Priority is 0. The highest most expedited priority is 9. The default value is 4.
Put Timeout	Time, in milliseconds, to wait for the queue to become available when attempting to put a message on the queue. You can enter 0 for an unlimited wait, but this is not recommended. If no value is supplied, timeout will be set to 3000 milliseconds.
Request Context Namespace	For synchronous or asynchronous processing, namespace or list of namespaces containing registers that will be made available to the Internal Queue listener. Select *Default Namespace* for all registers in default (no prefix) or *None* to send no registers at all. Enter an asterisk (*) for registers from all namespaces. The default value is default.
Response Context Namespace	In synchronous processing, registers set by Internal Queue listener in the *return* namespace will be made available to the calling process. If a value is supplied here, then response registers will be copied to the specified namespace. Leave this parameter blank to store response registers in the *return* namespace.

Parameter	Description
Return (required)	Select one of the following options from the drop-down list: ❏ **status.** The status document will be the output document. This is the default value. ❏ **input.** The input document will become the output document. ❏ **response.** Awaits synchronous output from the Internal Queue listener. **Note:** The response option is not compatible with local transactions.
Timeout	Determines how many seconds to wait for synchronous response. Set to 0 or leave blank to wait indefinitely. The default value is 0.

Reference: Internal Emit Service Edges

The following table lists and describes the edges that are returned by the Internal emit service.

Edge	Description
OnSuccess	Operation was successful.
OnFailure	Fail condition occurred during execution.
OnError	Exception occurred during execution.
OnParseError	Could not parse a document.
OnNotFound	Resource was not found. This may or may not be a failure.
OnTimeOut	Operation timed out.
OnCancelled	Service has responded to a cancellation request.
OnNotFound	Resource was not found and this is considered an error.

Edge	Description
OnFailedOperation	Could not perform the operation requested.

Chapter 3

Ordered Queue Processing

This section describes how to configure Ordered queue processing.

In this chapter:

Ordered Queue Processing Overview

A common application requirement is to present messages to a channel for processing in an order that is not directly related to the order of their arrival at the prior channel. For example:

- **Retail Applications.** EDI messages for shipping arrive from the warehouse in random order, and must be dispatched to the appropriate customer record for invoicing.
- **Medical Applications.** Test results are generated by a lab in the order of processing, but need to be dispatched to the record of a patent as a group (all or none).
- **Shipping Applications.** Messages must be sent to a remote location in processing order. If the entire message set cannot be created, then none of the messages should be sent.
- **Cross-system Message Exchange Application.** Sequenced messages are sent across the web to another system. The arrival order at the destination cannot be predicted, and the messages must be processed in a particular sequence that is not related to their arrival time.
- **Unordered batches must be processed in a specific order.** Incoming messages, perhaps from an iterator over an incoming message, must process portions of a message in a sequence that is different from the defined order of the elements in the original message.

Underlying these types of requirements is a system that receives in some intermixed sequence, is charged with not losing any messages, and desires to process the incoming messages in parallel. Parallelism loses the input ordering. However, single threading has an adverse impact on performance.

The Ordered listener is an iWay Service Manager (iSM) channel that alleviates these concerns of parallelism, retaining the required order, and collecting messages for a specific purpose for dispatch to the next channel. Messages can be dispatched as they become available, while retaining the order, or as an ordered group when the final message of the group is recognized. With the exception of the ordering facility, an ordered queue is similar to the commonly used internal queue, and it retains the characteristics of that type of channel. This includes support for low and high watermark tracking, passivation, inhibition, and so on.

It is the responsibility of the application designer to determine the type of queue required and to place messages on the queue appropriately for the application.

Introducing the Ordered Queue Facility

The ordered queue facility of iSM collects messages for dispatch and ensures that they are dispatched in the appropriate order. It is a variation of the Internal Queue listener and emit service, and supports the complete set of capabilities of the internal queuing facility.

In the internal queuing facility, messages are emitted to the queue, where they are selected for execution generally in the order of arrival. Multiple subchannels can be defined to process the messages in parallel. Pended messages are placed back on the queue as they are selected to be retried. The use of multiple subchannels (threads) and the use of pending prevent any control of message ordering. To maintain ordering, applications frequently define a single subchannel and avoid use of the pending facility. This technique maintains ordering, but at the expense of performance and complexity. This is the issue that the ordered internal queue facility is intended to address.

The ordered queue facility allows messages to be categorized by group. A group is any value of meaning to the application. For example, a patient ID, a transaction ID, a shipment number, and so on. The group identifier is passed along with the message from the emit service. Messages within the group will be dispatched by the Ordered Queue listener in the order they are received. The group is defined when it is first presented for a message. There is no need to predefine the group during configuration.

Within a group, subsorting by a designated sequencing field is possible, allowing the ordered queue to resort the messages before passing them to the application process flow. This allows situations, such as the unpredictability of input order due to communication delays for some messages or unsorted input in general to be handled. The intra-batch subsort key is presented by the emit service. For non-batching queues, subsorting is not available.

Configuring an Ordered Queue Listener

The Ordered Queue listener is used to manage an ordered queue. In this section, the term queue refers to the entire set of messages that are awaiting transfer to a subchannel.

The ordered queue is a collection of queuelets. Each queuelet collects the messages of one group in the order they were received. A new queuelet is created automatically when a message comes in with a key that has no associated queuelet. A queuelet is automatically deleted when it becomes empty, that is, when all the messages in that group have completed execution. Messages are deleted from the queuelet after their execution is completed. This ensures that messages cannot be lost when the queue is configured to be persistent, and maintains the ordering should a message arrive for the group while execution of the last message is proceeding.

This section will address differences in configuration between the simpler internal queue and the ordered queue. For more information on how the ordered queue system works for the ordering of messages, see *Introducing the Ordered Queue Facility* on page 28.

To configure an Ordered Queue listener, you must create a channel for your application project using the Channel Builder in iWay Integration Tools (iIT) and select *Ordered Queue* as the listener type for your inlet. For more information, see *Configuring a Listener Using iWay Integration Tools* on page 81.

For a complete description of the configuration parameters that are available for the Ordered Queue listener, see *Ordered Queue Listener Configuration Parameters* on page 34.

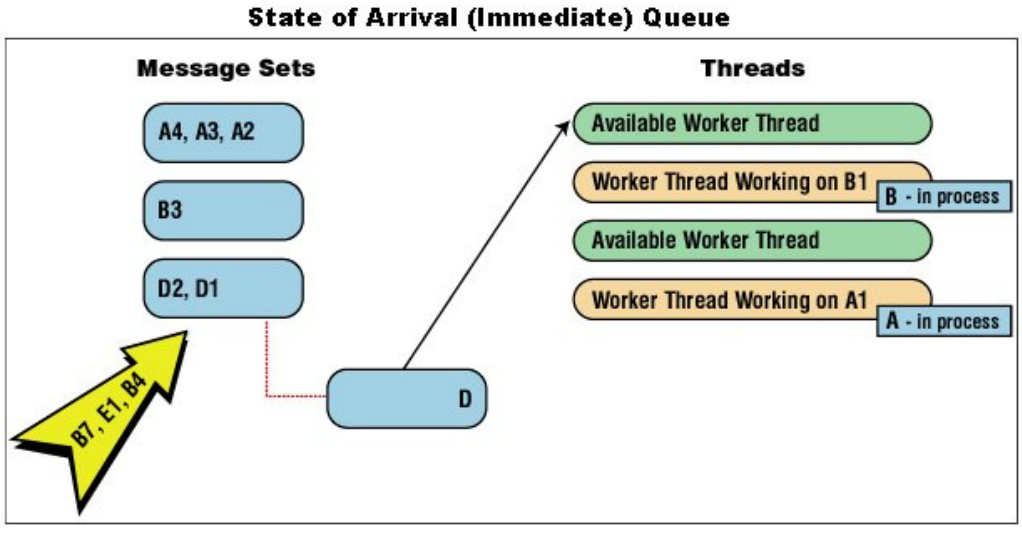
For a complete description of the Ordered Queue listener Special Registers (SREGs), see *Ordered Queue Listener Special Registers* on page 38.

Pended Messages

Pended messages are always executed in the proper order within the group. The implication is that it is the group itself that is pended rather than the individual message. Consistent with the normal server rules, if the pended message times out or fails, then it is considered to be complete and the next message in the queuelet is made available for dispatch.

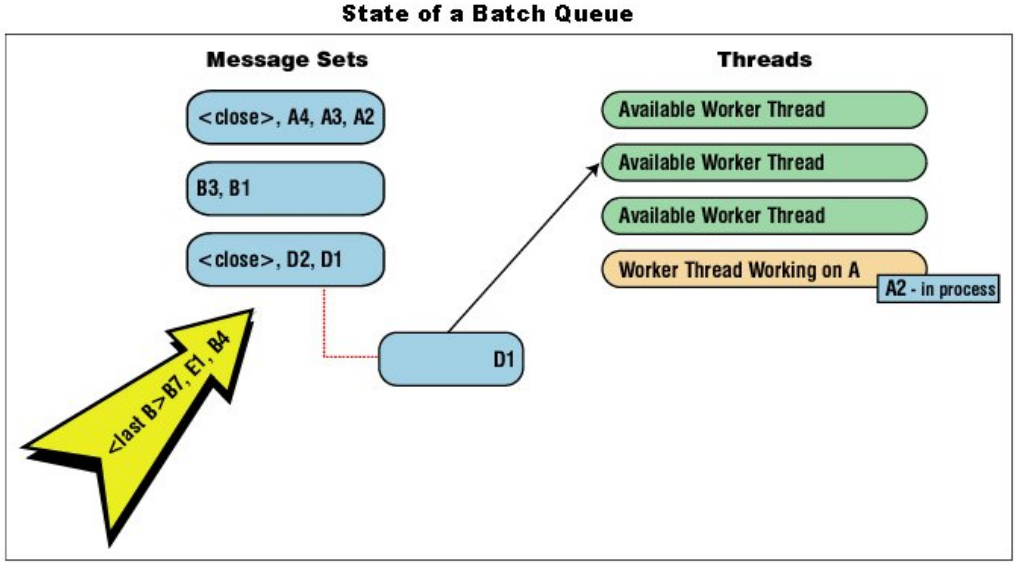
Immediate Mode Queues

In immediate mode, the messages are available for dispatch to the subchannel (process flow) immediately upon their arrival at the listener.



Batch Mode Queues

In batching mode, a new message is put in its queuelet. However, the queuelet simply collects the messages without making them available for dispatch. When the end of group signal is received for that group, the queuelet becomes available for execution, just like in immediate mode.



A delete group message can be sent to delete an unended group. A delete message cannot be used for an ended group.

In persistent mode, the end of group signal is pended along with the messages, thus instructing the channel to correctly end the group when reloading it from safe store. An unended group is reloaded as an unended group. Application logic must decide whether to end that group or delete it.

Control signals, such as delete group, are sent through the Ordered emit service (com.ibi.agents.XDOrderedEmitAgent), which is configured using the Queue Out object in the iWay Process Flow Designer and selecting the *iWay queue (ordered) - send a message* action. Available signals are listed and described in the following table:

Signal	Description
Document {doc}	Current document message is to be enqueued for its group key.
Delete {del}	Batch queue is to be deleted.
End Signal {end}	Final message has been enqueued for this group, and the batch queue is to be made available for dispatching.
Keep Alive {keepalive}	Resets the timeout period for a batch queue.
Last Document {docend}	Effects a document and end operation. Signals that this message is the final message and the queue is to be made available for dispatching.

The following table lists and describes the general set of special registers (SREGs):

Special Register	Description
endofgroup	Set to *true* when the last message of a batching queuelet is dispatched to the process flow. This register is set to *true* for this final message. Emitted signals, such as the end of group signal are not dispatched to the associated process flow.
msgkey	Group key associated with this message.
sortkey	Sort key for non-immediate processing. Not applicable for non-sorted immediate processing.

The ordered listener has a parameter to define the timeout before a batching group must be closed. This is the maximum time between the last message of a group and receipt of the end of the group signal. The timeout does not apply to queues in immediate mode. The Expired Group Event Flow parameter holds the name of a process flow to run when the group timeout expires. If no process flow is specified, then the server writes an error message in the log. The messages are enqueued to the dead letter destination (assuming one is configured) and the group is deleted. If a process flow is specified, then the server runs that process flow. The process flow can send new messages, the delete group message, the end of group signal, or the *keep alive* signal to reset the timer for the specified group. If the process flow returns a success condition, then the server considers the condition to be handled, and no further action is taken. If the process flow returns a condition other than success, then the server reverts to the default behavior as if there were no process flow.

The expired group flow receives a signal message in the event of the timeout expiration. This flow can elect to emit a *keep alive* or a *delete* signal message to the group.

```
<signal errors="0" protocol="ordered" timestamp="2018-10-24T17:56:30.730Z"
type="expired" version="2">
    <parms count="2">
      <parm name="queue">ordered</parm>
      <parm name="key">patient1</parm>
    </parms>
</signal>
```

The type of the signal document is *expired* and specifies the queue and group names, which in this example are *ordered* and *patient1*.

Stopping the Server

If the channel in immediate mode is stopped, then the messages behave as defined for the Internal Queue listener.

- A persistent queue is stopped immediately upon completion of any messages that are in process. The persisted messages are available once the server is restarted.
- A non-persistent queue will complete messages that are in the queue before stopping the channel.

Restarting the Server

When the server starts, it searches for persisted messages awaiting execution. The queuelets are reconstructed to their status prior to the when the server was stopped. Immediate mode and ended batch mode queuelets begin dispatching messages immediately, while unended batch mode queuelets wait for an end of group signal.

Application designers are reminded that the timeouts applied to batch mode queues are based on wall clock time. This can result in the execution of timeout flows, and the application logic should take this into account.

Reference: **Ordered Queue Listener Configuration Parameters**

The following table lists and describes parameters for the Ordered Queue listener.

Note: Parameters that are common to queue listeners are described in *Listener Configuration Parameters* on page 95.

Property	Description
Name of Ordered Queue (required)	Name of the ordered queue that is used to identify the ordered message listener destination.
Queuing Mode	Determines how received messages are handled. Available modes include: ❑ **immediate.** As messages for a group are received, they become available for dispatching. ❑ **batch.** A message becomes available for dispatching only when the group is closed. The default value is immediate.
Sorting Mode	Applies to batch queues only, and only within the batch itself. Available options include: ❑ **Chronological** (default). No intra-batch sorting is performed, and the order of presentation to the application is arrival order. ❑ **Lexical.** The subsort key is sorted in lexical order. For strings, this is generally considered alphabetic order. ❑ **Numerical.** The subsort key is sorted numerically by value rather than as a string. The default value is Chronological.

Property	Description
Group Timeout	Indicates the time that a batching queue can remain unavailable for dispatching following the receipt of the latest message. In effect, this is an inactivity timeout. The default value is 300 seconds.
Expired Group Flow	Name of a published process flow that receives control if an unended group times out.
Persistent	If enabled, messages are persisted. Persistent messages are held in the safestore until completion and can be recovered if the server is restarted. Select one of the following options from the drop-down list: ❑ none {false} ❑ rdbms {rdbms} ❑ file {true} The default value is none {false}. The *rdbms* option enables the application to persist messages to a remote database location where, in the event of a failover situation, a secondary server running remotely can continue processing persisted messages without any business interruption. To use RDBMS persistence, you must create the iway_queues table in the JDBC provider, using the DDL script in the etc/setup directory. A hot backup channel can be configured to use the same RDBMS safestore.
Safestore Location	If persistent, this is the location in the file system to which documents are safestored.

Property	Description
Compress Persistent	If the queue is persistent, the documents written to the safestore can be compressed. Select one of the following options from the drop-down list: ❑ smallest {best} ❑ none {none} ❑ fastest {speed} ❑ standard {std} The default option is to not compress (none).
Low Mark	If the size of the named queue falls below this value, then the named listeners that are specified in the Control List parameter are sent an activate message. The low mark value applies to the queue as a whole, and not to the individual quelelets. The default value is 0.
High Mark	If the size of the named queue goes above this value, then a passivate message is sent to the listeners that are specified in the Control List parameter. The high mark value applies to the queue as a whole, and not to the individual quelelets. The default value is 0.
Control List	If a high or low mark value is crossed, the appropriate message is sent to each listed listener. For example, if two listeners LISA and LISB are feeding the internal queue, listing them as LISA,LISB will cause each to receive the appropriate message.

Property	Description
Inhibit Add	If set, the queue will not accept new messages when its size reaches the high mark, and will resume accepting messages when the number of messages on the queue reaches the low mark. The effect of this inhibition can cascade through the application, controlling overall performance. The default value is false.
External Mark	When set to a number greater than 0, this causes the message to be stored in memory up to the specified number. When the queue grows larger than the specified number, the message is stored on disk. This is equivalent to splitting the memory and performance optimization for the messages in the queue, and is effective for large lists that may fill memory. The default value is 0.
Duration	Maximum time in seconds (allows xhxxmxxs format) that a document can remain in this channel, starting from when the document is added to the queue, including pends and retries. The default value is 24 hours.
Support Pending	If set, messages can be pended for later execution if the process flow calls for a Fail/Pending operation. Pending messages persist for a specified time and are retried at a specific interval. You might use a pending operation in the event that a message cannot be processed because an external resource is currently unavailable. The default value is false.
Retry Interval	Determines the interval (in seconds) between retrying pending requests. The default value is 600 seconds (10 minutes).

Reference: Ordered Queue Listener Special Registers

The following table lists and describes the Special Registers (SREGs) available on the Ordered Queue listener. These values can be used in the application for message routing and processing.

Name	Source	Level	Type	Description
endofgroup	Listener	Document	Boolean	Indicates the last message of a queuelet group.
iway.channel	Listener	System	String	Full name of the channel (may include channelname.inlet.listener).
iway.channelname	Listener	System	String	Channel name portion of the name from the full channel name of channelname.inlet.listener.
iway.inletname	Listener	System	String	Inlet name portion of the name from the full channel name of channelname.inletname.listener
iway.listener	Listener	System	String	Name of the listener.
iway.pid	System	System	String	Process ID of the server, if available.
iway.serverfullhost	System	System	String	Full host name of the server (includes domain).
iway.serverhost	System	System	String	Host name of the server.
iwayconfig	System	System	String	Current active configuration name.
iwayhome	System	System	String	Base at which the server is loaded.
iwayversion	System	System	String	Release version of the server.

Name	Source	Level	Type	Description
iwayworkdir	System	System	String	Path to the base of the current configuration.
msgkey	Listener	Document	String	Batch key being dispatched.
msgsize	Listener	Document	Integer	Physical length of the message payload.
name	Listener	System	String	Assigned name of the master (same as iway.channel).
protocol	Listener	System	String	Protocol on which the message was received.
sortkey	Listener	Document	String	Values being sorted for ordered dispatch, if applicable.

Configuring an Ordered Emit Service

The Ordered emit service is used to send a message to a group that is managed by an ordered queue. The document is marshaled with its context and then placed on the queue to be executed by the Ordered Queue listener and channel. The characteristics of message management and sequencing are controlled by the Ordered Queue listener and channel. The use of ordered queues is similar to the use of internal queues, which provide no ordering. For more information, see *Internal Queue Processing* on page 11.

To configure an Ordered emit service, you must create a process flow for your application project using iWay Integration Tools (iIT) and use the Queue Out object to emit to the Ordered Queue (iWay queue (ordered) - send a message). Note that the Queue Out object implements Ordered Queue operations using *com.ibi.agents.XDOrderedEmitAgent*.

For a complete description of the configuration parameters that are available for the Ordered emit service, see *Ordered Emit Service Parameters* on page 47.

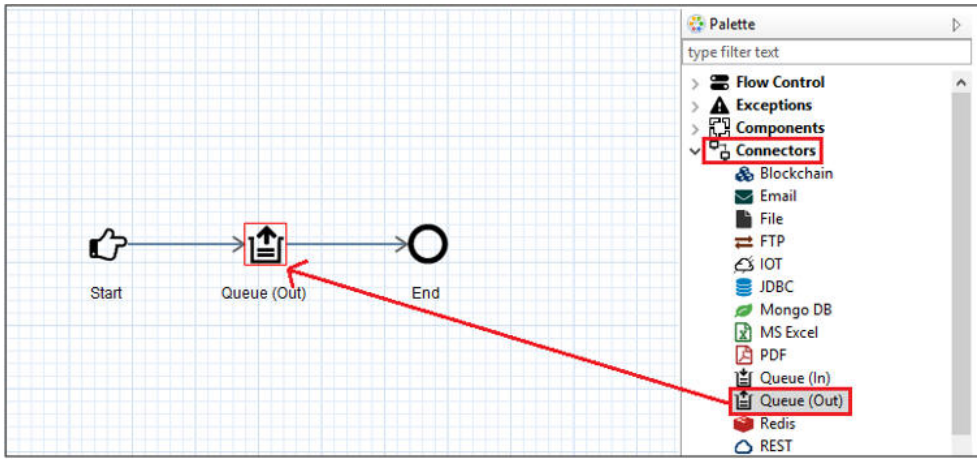
For a complete description of the edges that are returned by the Ordered emit service, see *Ordered Emit Service Edges* on page 49.

Procedure: How to Configure an Ordered Emit Service

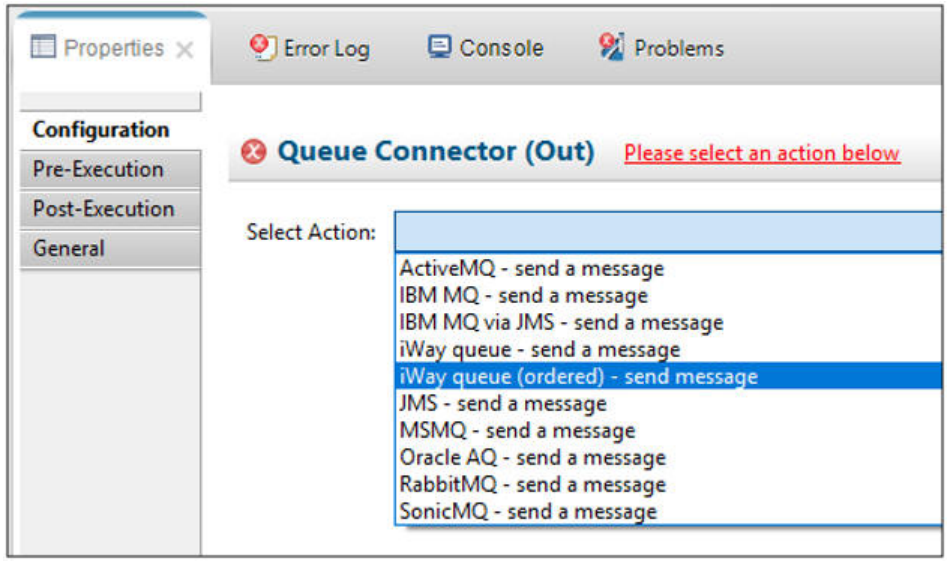
To configure an Ordered emit service, you must create a process flow in your application project using iWay Integration Tools (iIT) and use the Queue (Out) object from the Palette, under Connectors, to emit to the ordered queue (*iWay queue (ordered) - send message*) action.

Note: The Queue Out object implements the *iWay queue (ordered) - send message* action using the Ordered emit service (com.ibi.agents.XDOrderedEmitAgent).

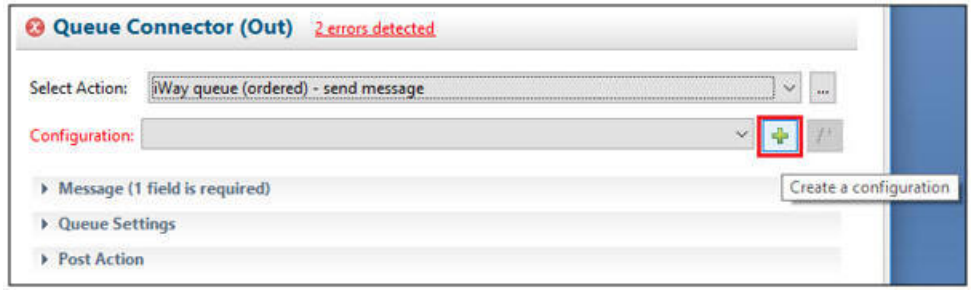
1. Expand the *Connectors* category in the Palette and drag the *Queue (Out)* object to your process flow, as shown in the following image.



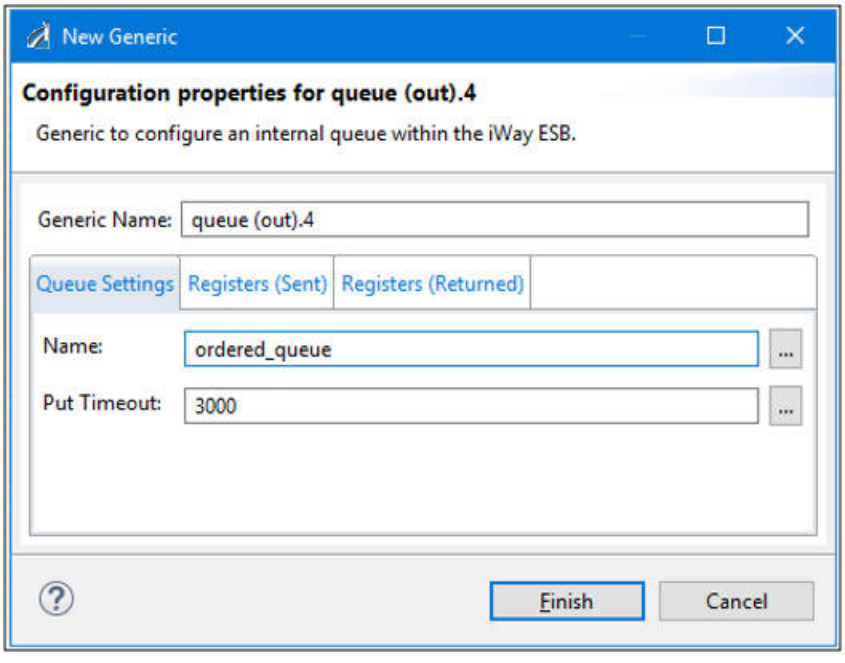
2. In the Properties tab, under Configuration, select *iWay queue (ordered) - send message* from the Select Action drop-down list, as shown in the following image.



3. Click the *Create a configuration* icon to the right of the Configuration field to configure a new generic for this object, as shown in the following image.



The New Generic dialog box opens, as shown in the following image.

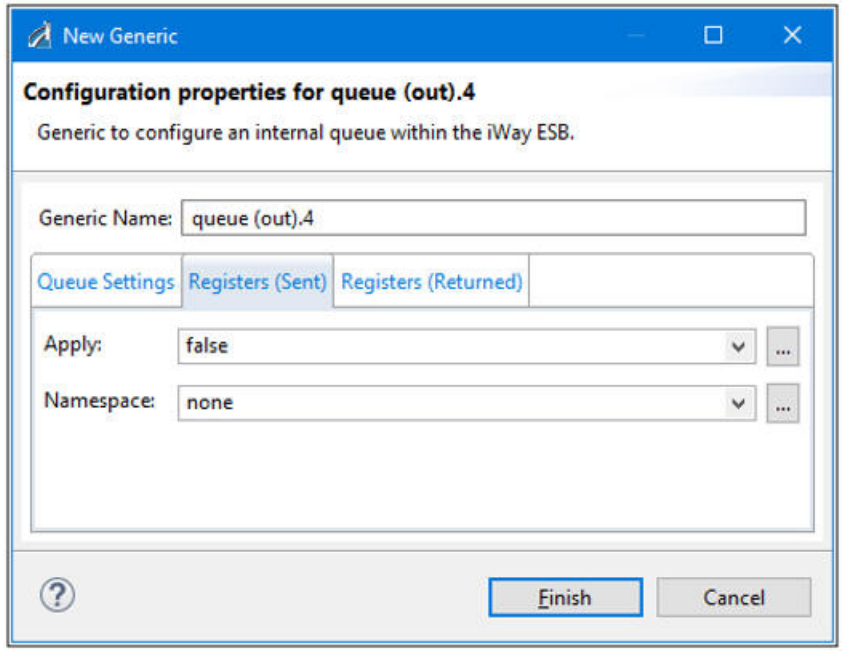


4. Specify a name for this generic in the Generic Name field, or accept the default.
5. In the Queue Settings tab, specify values for the following configuration parameters:

   - **Name.** Name of the ordered queue that is serviced by an Ordered Queue listener. The queue is created when the channel is started and exists as long as the server is running.

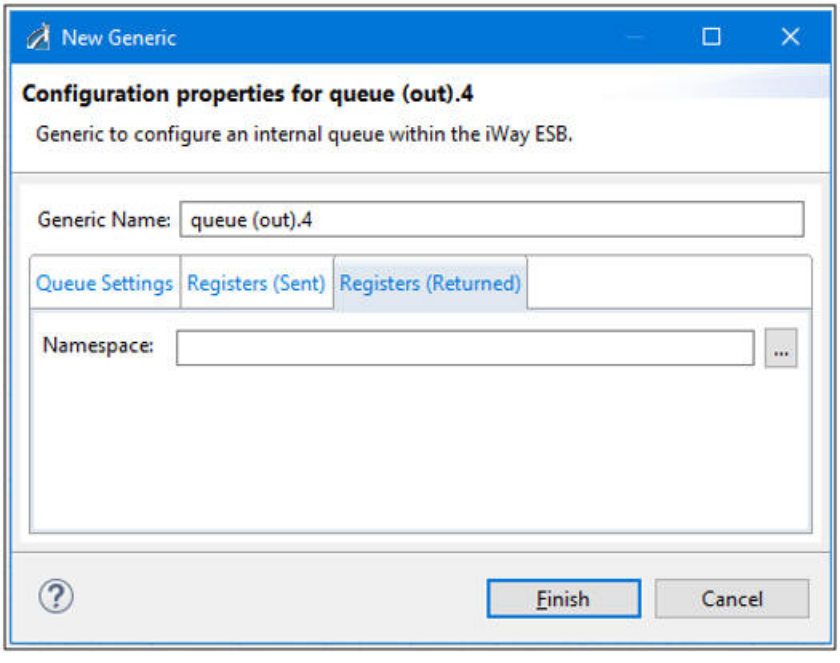
   - **Put Timeout.** Determines the amount of time that the emit will wait for the ordered channel to accept the message. If the ordered queue is inhibited, the emit is paused until the message can be accepted. If the timeout period expires, a status message is sent down the timeout edge, where your application might chose to pend the message. For more information on using inhibition to provide cascading flow congestion management, see the *Introducing iWay Service Manager* chapter in the *iWay Service Manager User's Guide*. If no value is supplied, timeout is set to 3000 milliseconds.

The Registers (Sent) tab is shown in the following image.



By default, the Apply parameter is set to *false*. If set to *true*, user-type registers are passed to the ordered queue. DOC and HDR registers are always transferred with the message. By specifying a value for the Namespace parameter in this tab, registers in that namespace will be sent to the ordered queue. This is used to limit the registers to those of interest to the message process.

The Registers (Returned) tab is shown in the following image.

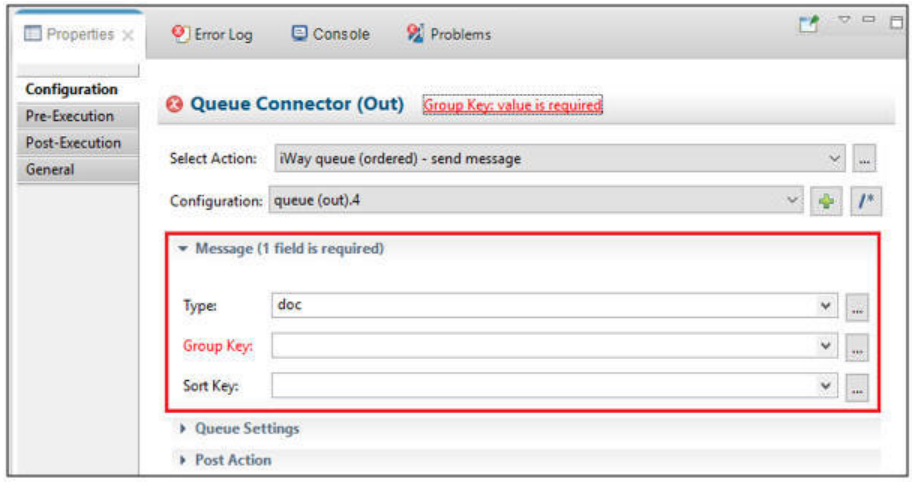


The Namespace parameter in this tab is used only for synchronous emits. If set, registers returned from the ordered channel are placed into the specified namespace.

6. Click *Finish*.

   You are returned to the Properties tab.

7. Expand *Message*, as shown in the following image.



8. Specify values for the following configuration parameters:

- **Type.** Classification of the message that is being sent to the ordered queue. Select one of the following message types from the drop-down list:
  - **Delete {del}.** The batch queue is to be deleted. Has no effect for an immediate queue.
  - **Document {doc}.** The current document message is to be enqueued for its group key. This is the default value.
  - **Last Document {docend}.** Affects a document and end operation. Signals that this message is the final message and the queue is to be made available for dispatching.
  - **End Signal {end}.** The final message has been enqueued for this group, and the batch queue is to be made available for dispatching. Has no effect for an immediate queue.
  - **Keep Alive {keepalive}.** Resets the timeout period for a batch queue. Has no effect for an immediate queue.
- **Group Key.** Identifies the group for this message. Groups are created in the ordered queue when a new key is presented, and are deleted when the last message for that group has completed execution.

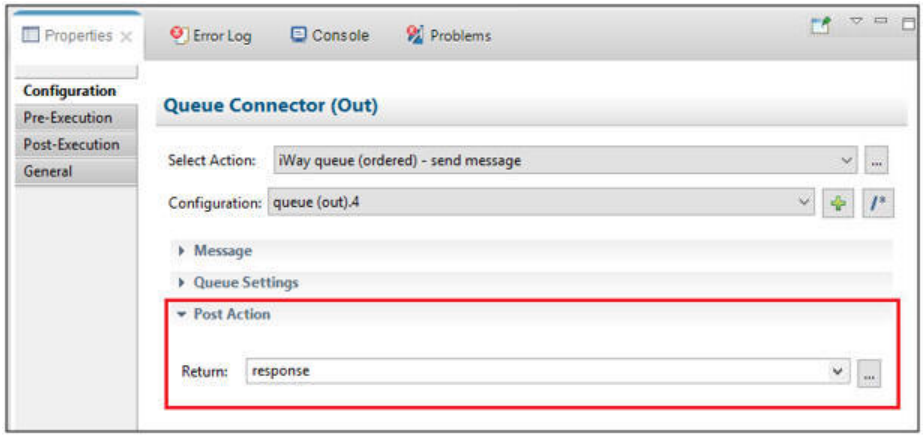
- **Sort Key.** Applied only for non-chronological batching queues. Passes the key to be used for the intra-batch sorting. Usually this will be an iFL statement extracting some value from the message itself, such as a sequence number.

9. Expand *Queue Settings*, as shown in the following image.

10. Specify a value for the following configuration parameter:

- **Respect Transactionality.** Determines whether to respect existing transactionality. The default value is true.

11. Expand *Post Action*, as shown in the following image.



12. Specify a value for the following configuration parameter:

- **Return.** Select one of the following options from the drop-down list:
  - **response.** The response document will be the output document. This is the default value.

- **status.** A status document is returned showing the success of the queuing operation.
- **input.** The input document that originally came into the emit service is returned.

13. Save your process flow.

Reference: **Ordered Emit Service Parameters**

The following table lists and describes parameters for the Ordered emit service.

Note: Parameters that are common to emit services are described in *Service Configuration Parameters* on page 99.

Parameter	Description
Queue Name (required)	Name of the ordered queue that is serviced by an Ordered Queue listener. The queue is created when the channel is started and exists as long as the server is running.
Group Key (required)	Identifies the group for this message. Groups are created in the ordered queue when a new key is presented, and are deleted when the last message for that group has completed execution.

Parameter	Description
Message Type	Classification of the message that is being sent to the ordered queue. Select one of the following message types from the drop-down list: ❑ **Delete {del}.** The batch queue is to be deleted. Has no effect for an immediate queue. ❑ **Document {doc}.** The current document message is to be enqueued for its group key. This is the default value. ❑ **Last Document {docend}.** Affects a document and end operation. Signals that this message is the final message and the queue is to be made available for dispatching. ❑ **End Signal {end}.** The final message has been enqueued for this group, and the batch queue is to be made available for dispatching. Has no effect for an immediate queue. ❑ **Keep Alive {keepalive}.** Resets the timeout period for a batch queue. Has no effect for an immediate queue.
Sort Key	Applied only for non-chronological batching queues. Passes the key to be used for the intra-batch sorting. Usually this will be an iFL statement extracting some value from the message itself, such as a sequence number.
Want User Registers	If set to *true*, user-type registers are passed to the ordered queue. DOC and HDR registers are always transferred with the message. The default value is false.

Parameter	Description
Put Timeout	Determines the amount of time that the emit will wait for the ordered channel to accept the message. If the ordered queue is inhibited, the emit is paused until the message can be accepted. If the timeout period expires, a status message is sent down the timeout edge, where your application might chose to pend the message. For more information on using inhibition to provide cascading flow congestion management, see the *Introducing iWay Service Manager* chapter in the *iWay Service Manager User's Guide*. If no value is supplied, timeout is set to 3000 milliseconds.
Request Context Namespace	By specifying a namespace, registers in that namespace will be sent to the ordered queue. This is used to limit the registers to those of interest to the message process.
Return (required)	Select one of the following options from the drop-down list: ❑ **status.** A status document is returned showing the success of the queuing operation. ❑ **input.** The input document that originally came into the emit service is returned. The default value is status.
Response Context Namespace	This parameter is used only for synchronous emits. If set, registers returned from the ordered channel are placed into this namespace.

Reference: Ordered Emit Service Edges

The following table lists and describes the edges that are returned by the Ordered emit service.

Edge	Description
OnSuccess	Operation was successful.

Edge	Description
OnFailure	Fail condition occurred during execution.
OnError	Exception occurred during execution.
OnParseError	Could not parse a document.
OnNotFound	Resource was not found. This may or may not be a failure.
OnTimeOut	Operation timed out.
OnCancelled	Service has responded to a cancellation request.
OnNotFound	Resource was not found and this is considered an error.
OnFailedOperation	Could not perform the operation requested.

Chapter 4

Reverse Invocation Queue Processing

This section describes how to configure Reverse Invocation (RVI) queue processing.

In this chapter:

Reverse Invocation Queue Processing Overview

Reverse Invocation (RVI) queue (also referred to as gateway) processing links two or more iWay Service Manager (iSM) instances in a message receiver or a message executor relationship to tunnel through secure firewalls.

To configure RVI queue (gateway) processing, you must:

1. Install the iWay Gateway extension on the iWay Proxy server and the execution engine.

   To install the iWay RVI Proxy, you must add the Gateway extension to your iSM instance during the iSM installation. For more information on installing iSM, see the *iWay Installation and Configuration Guide*.

   After the Gateway extension is installed, the RVIAttach listener, RVIGateway listener, and RVIRelay service are added to the design-time registry and run time configurations.
2. Configure the RVIAttach listener on the iWay Proxy server.
3. Add the RVIRelay service to the appropriate listener(s) configured on the iWay Proxy server.
4. Configure the RVIGateway listener on the execution engine.

iSM horizontal scaling through reverse invocation allows a message received by one iSM configuration to be processed on another configuration. Configurations are expected to be on separate machines, but this is not a requirement. Messages can be distributed over an arbitrary number of associated configurations to balance workload and provide for high availability of processing services.

Messages are received at a receiving engine (the iWay Proxy) and executed at an execution engine. Each message arriving at the iWay Proxy is assigned to a named service. This assignment can be configured in a fixed manner based on the receiving listener or it can be assigned using the full services of iSM intelligent routing services. Regardless of how the assignment is made, the receiving engine locates an execution engine offering the named service, and passes the message to that engine for execution.

Processing engines connect to the receiving engine on a secure, reverse channel. This enables the receiving engine to be located across a firewall, enabling execution to be carried on in a secure environment not open to outside, unauthorized access.

This is also referred to as Reverse Invocation because the execution engine connects to the receiving engine rather than the receiving engine connecting to the execution engine to pass a document.

Proxy Service

Messages arrive at the proxy through any of the protocols that are supported by iSM. Each protocol is managed by a listener. The listener is configured to pass the message to a relay service, which selects an attached execution service and passes the message to the selected engine for execution. All other iSM capabilities are supported. For example, intelligent routing can examine the incoming message to select the appropriate relay service for execution.

Execution Service

The execution engine accepts relayed messages, executes them, and returns the result to the relay service, which in turn relays the result back to the configured emit service(s). Usually, ancillary emit operations are performed on the execution engines, though this is not required.

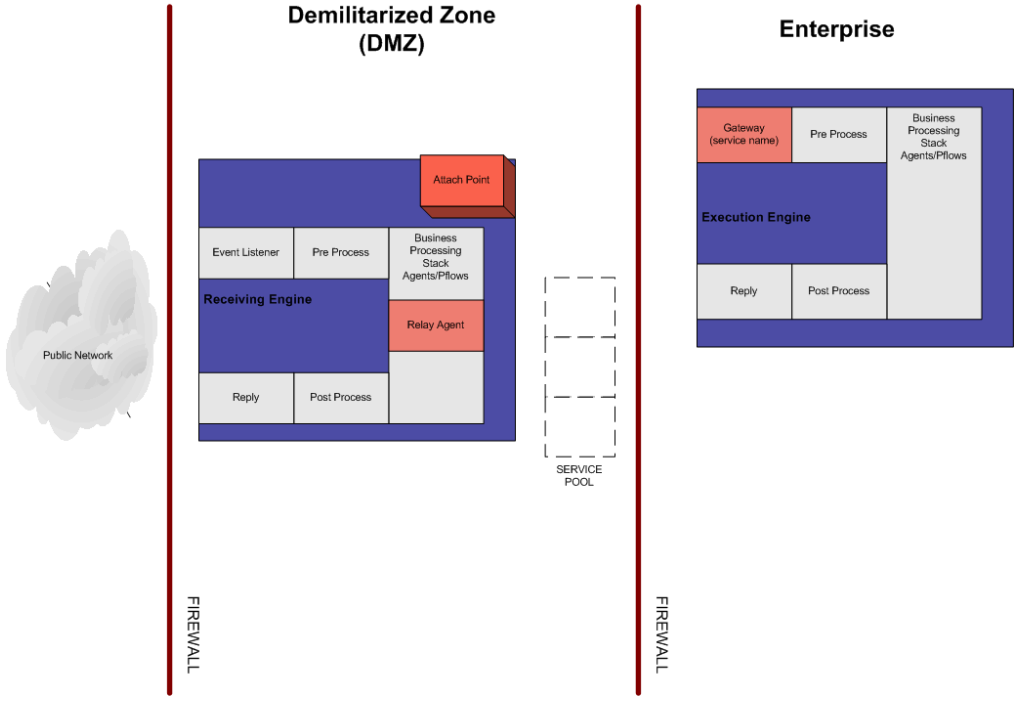
An execution engine is configured with one or more gateway listeners. A gateway is a named service that attaches to the attach point of a receiving engine. There must be one gateway for each service name offered, at each receiving engine attach point.

The process flow that is configured on the execution service must return only one result message. Although a process flow can be developed that returns multiple results, this practice is not compatible with the execution service.

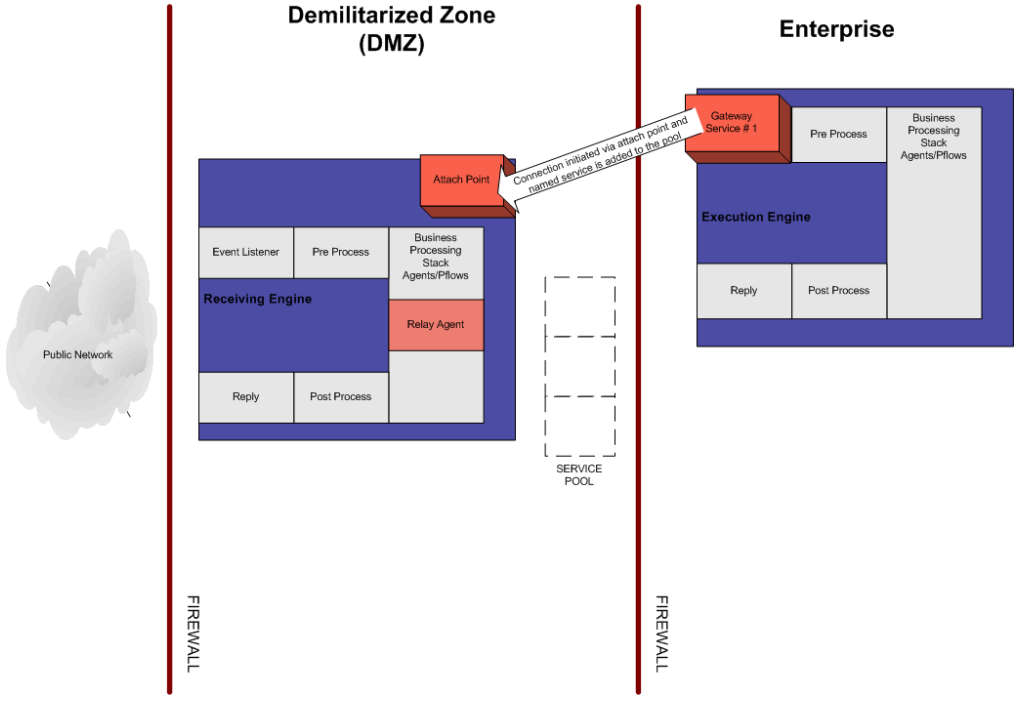
Reverse Invocation Process

This section depicts the reverse invocation process in a step-by-step fashion. In this depiction, iSM is deployed to two locations, one within the enterprise and one in the demilitarized zone (DMZ).

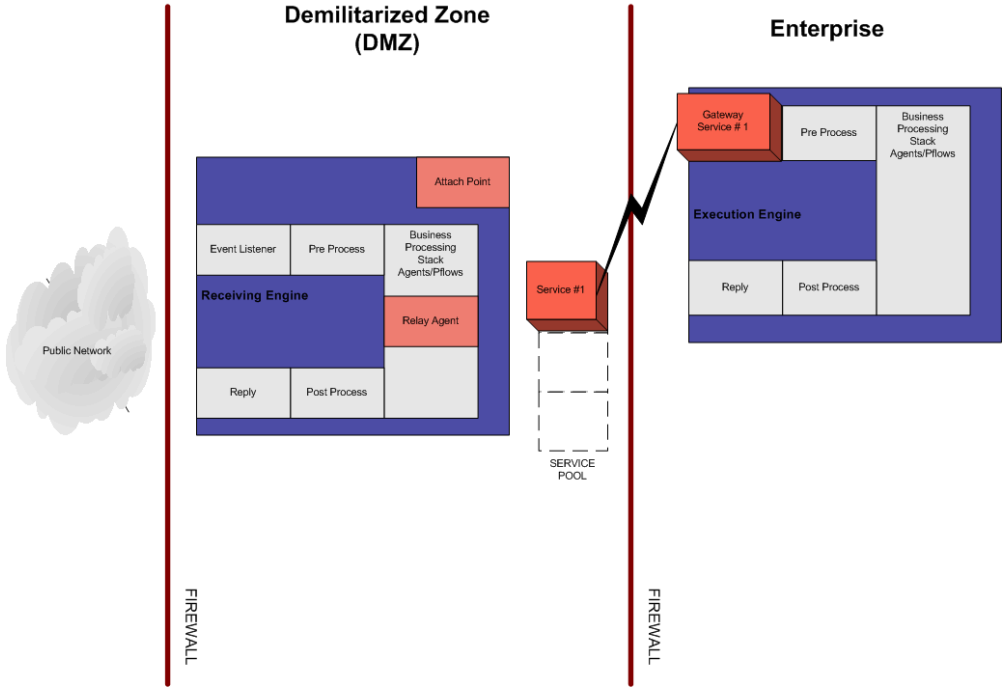
1. The iWay Proxy, or Receiving Engine, starts with the RVIAttach listener waiting for connections to be initiated from the Execution engine, as shown in the following image.



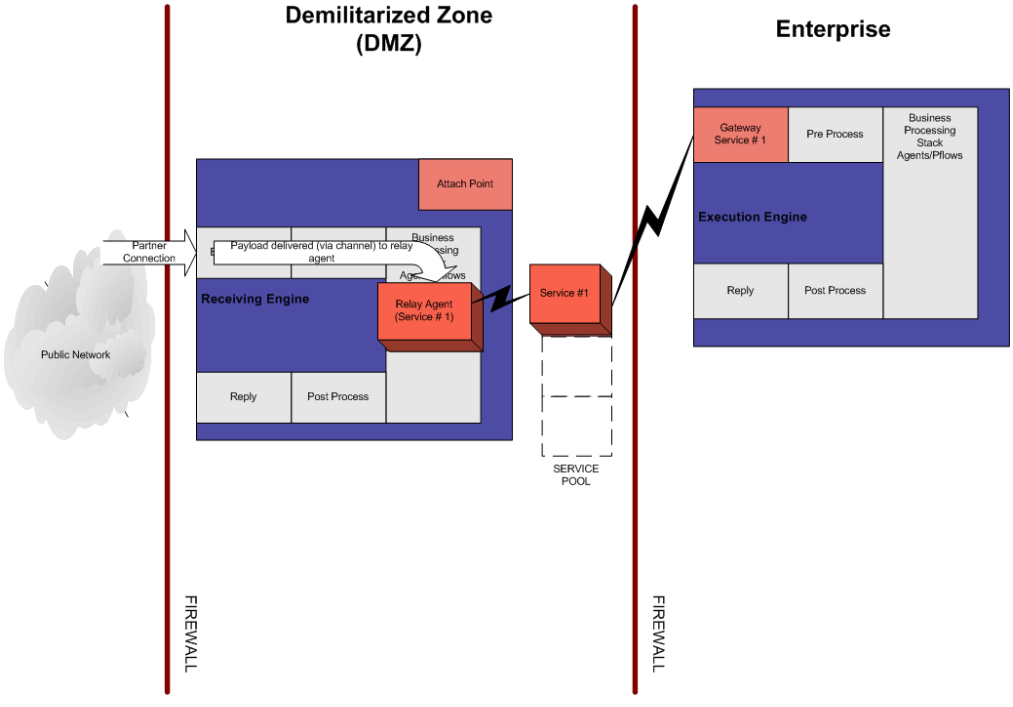
2. The connection is initiated by the gateway listener configured on the Execution engine located in the enterprise, behind the firewall. A service name is defined in the gateway listener configuration, as shown in the following image.



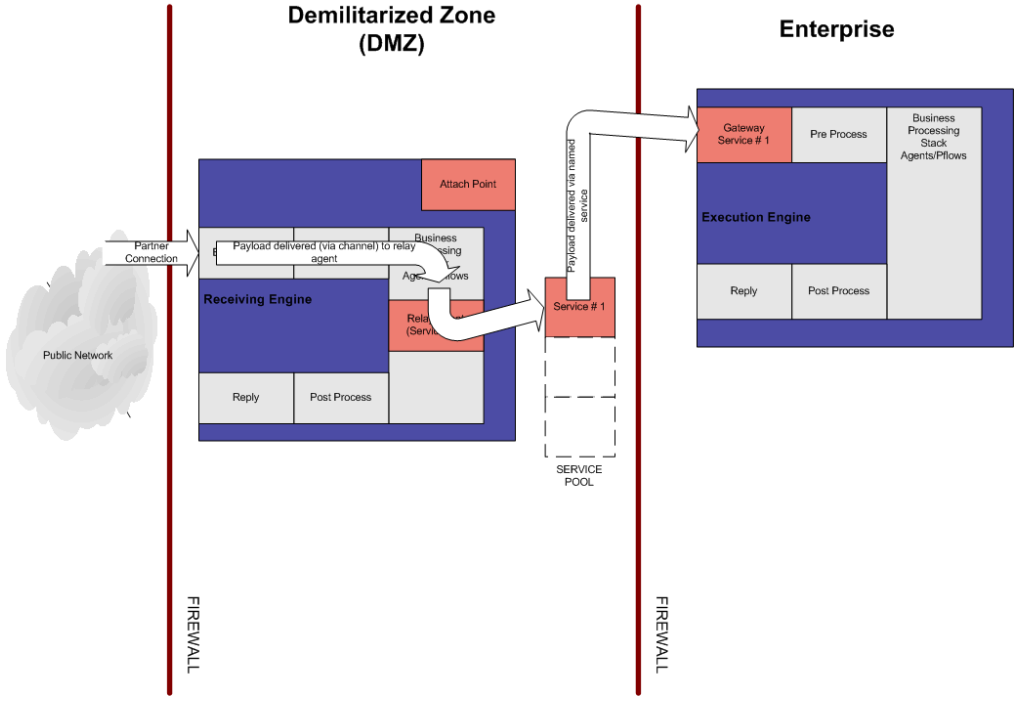
3. After the connection is established, it is added to a pool of connections and can be referenced by the service name, as shown in the following image.



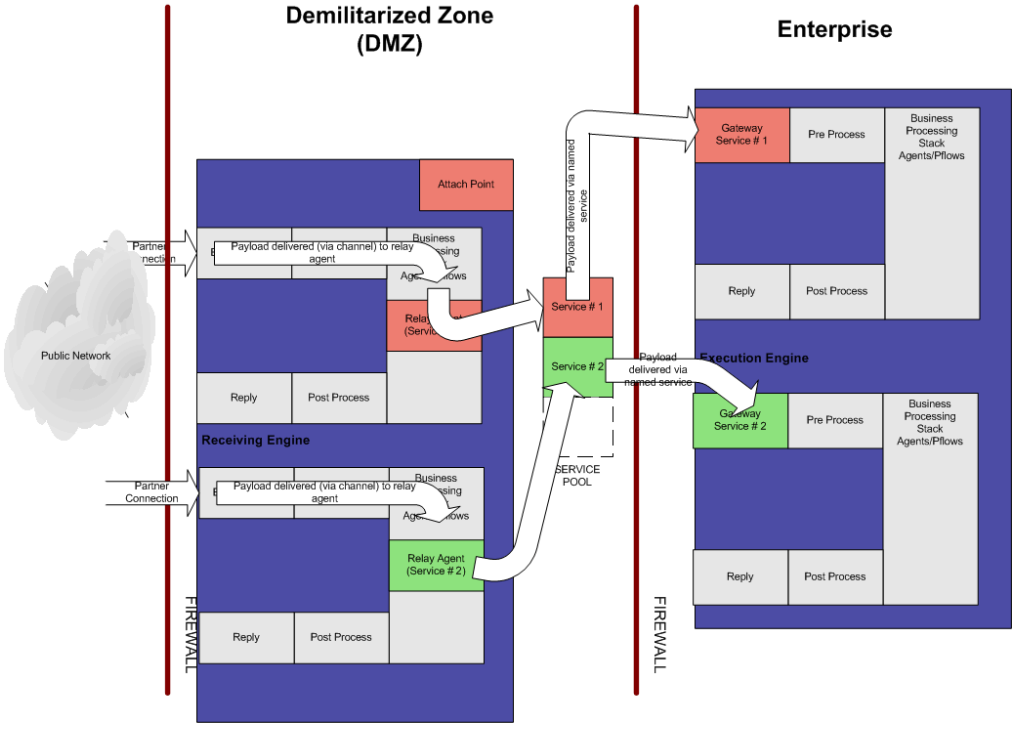
4. When a partner connects to the event listener defined on the iWay Proxy, the message is routed to the Execution engine through the relay service that is added to the event listener. The relay service is configured with the service name defined in the gateway listener configuration, as shown in the following image.



5. After the connection between the iWay Proxy and the Execution engine is established, messages pass securely through the configuration, as shown in the following image.



6. Multiple channels can be configured in the same way. Gateway listeners configured on the Execution engine can spawn services that the iWay Proxy can use to pass data to the configured gateway listeners, as shown in the following image.

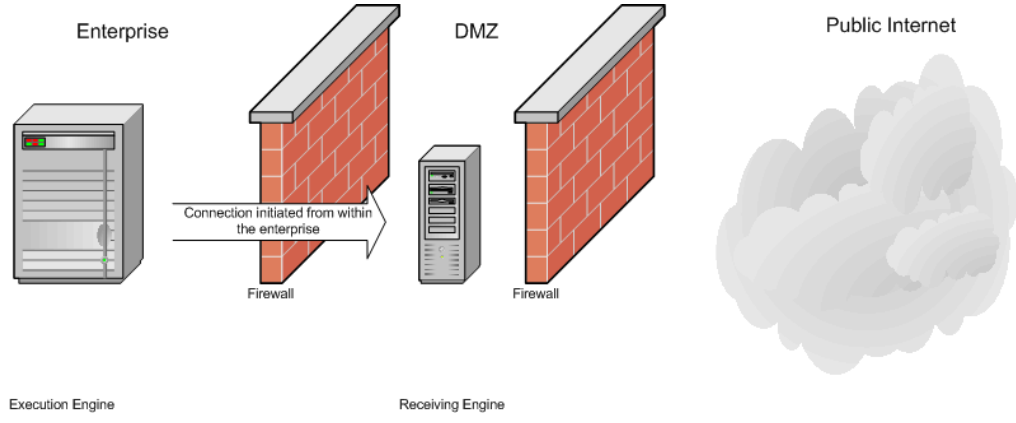


Sample Scenario

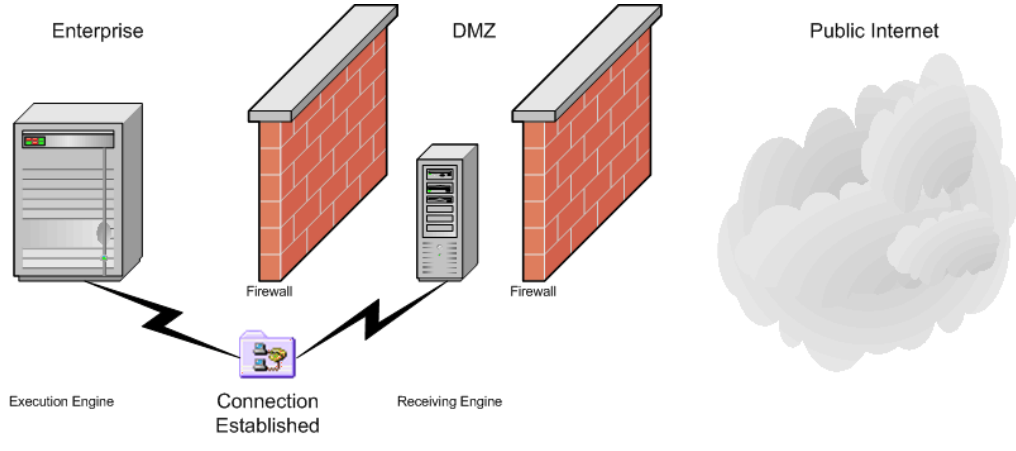
As an example of a Reverse Invocation scenario in which the payload is an EDI document, an AS2 message is routed over the public Internet. The message must be processed securely within the enterprise, where security certificates reside. The iWay Proxy server receives the message securely within the DMZ and passes it back for secure processing to an iSM located inside the enterprise that acts as the Execution engine.

The following diagrams depict the process:

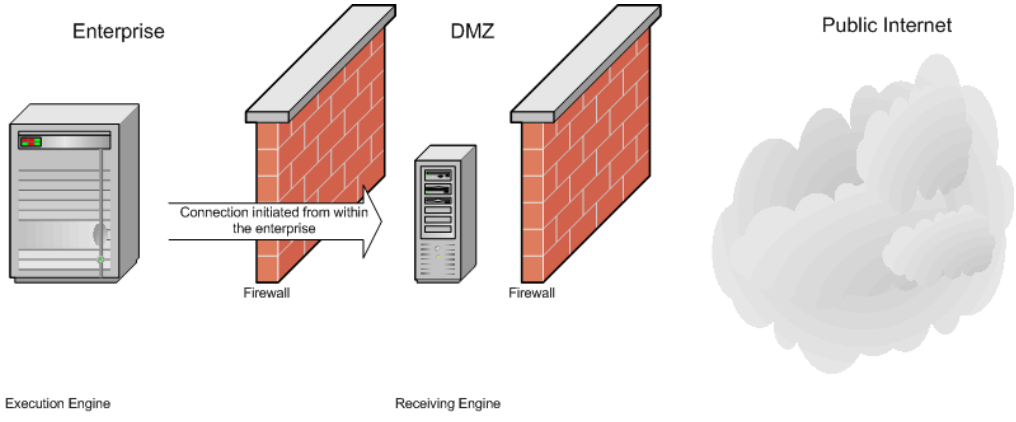
1. The Execution engine initiates a connection with the Receiving Engine (iWay Proxy).



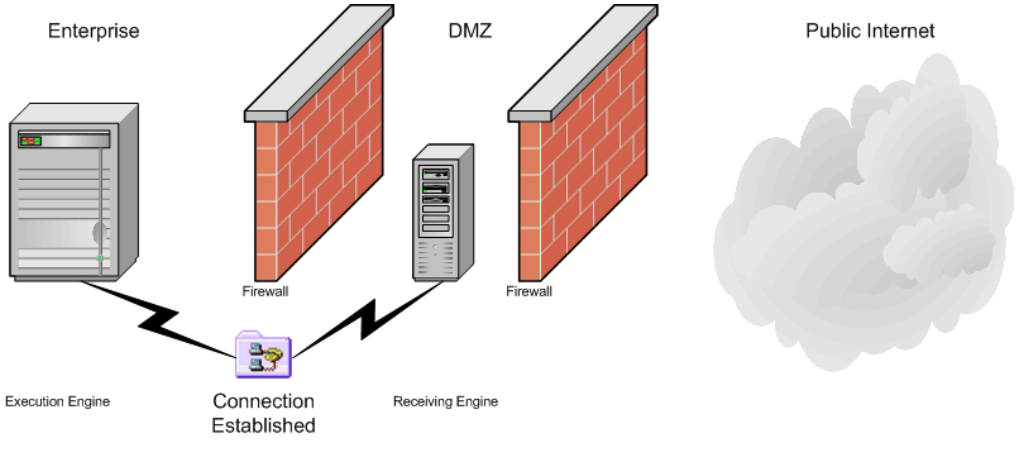
2. The session is established.



3. The trading partner initiates a connection with the iWay Proxy (the Receiving engine).



4. The connection is established, and the iWay Proxy manages connectivity between the trading partner and the internal processes hosted by the Execution engine.



From the perspective of a trading partner, a secure connection is established, and information can safely pass through the firewall for secure processing.

Configuring the RVIAttach Listener

Each Relay server maintains a list of attachment points, which is used to direct relayed messages to available Execution engines. Attachment points are characterized by a service name, an IP address, and a port. The Relay service channel is configured with the service name, which is resolved at run time to the target IP and port. Service names should be descriptive, but need not be related to a message type, channel name or host name. Multiple attach points for the same service name may be registered with one relay server, in which case, the connections they represent are assigned to relay events using algorithms which maintain a balanced work distribution (for example, Least Recently Used).

The purpose of the RVIAttach listener is to process attach messages from running Execution Channels in order to construct the attachment point list. The RVIAttach logic maintains the integrity of the attach point list by removing connections which have become unavailable. When that happens, other attach points offering the same named service are not affected.

To configure the RVIAttach listener, you must create a channel for your application project using the Channel Builder in iWay Integration Tools (iIT) and select *RVIAttach* as the listener type for your inlet. For more information, see *Configuring a Listener Using iWay Integration Tools* on page 81.

For a complete description of the configuration parameters that are available for the RVIAttach listener, see *RVIAttach Listener Configuration Parameters* on page 61.

Reference: RVIAttach Listener Configuration Parameters

The following table lists and describes parameters for the RVIAttach listener.

Property	Description
Port (required)	Port on which the attach listener is listening to receive service attachments.
Local Bind Address	On a server with multiple physical network interfaces, this specifies the interface to which the listener is bound. This can usually be left blank.
SSL Context Provider	Defined iWay Security Provider for SSL Context.

Property	Description
Allowable Clients	Optional host name or IP address, which, if entered, limits connections to those from the designated host or IP address. Only one host name or IP address is allowed per RVIAttach listener. If you wish to allow a set of Executor hosts to connect, one RVIAttach listener must be configured for each.
Timeout	Frequency with which the attach point checks for stop requests. The default value is 2 seconds.
Keep Alive	The interval at which to poll to ensure that a connection is still available. If an interval is specified, the attach point sends a keep alive message on each attached link. Care should be taken in setting this property, as overly short polling intervals can impact bandwidth and CPU utilization. The default value is 0; 60 seconds is recommended.

Configuring the RVI Relay Service

The RVI Relay service is responsible for passing messages to the Executor Server from a channel running on the Proxy Server. To accomplish this, the service uses its service name to find a matching attachment point in the attachment point list. If there are several matching attachment points, the system applies a load balancing algorithm to select which attachment point to use. The RVI Relay service may be configured with the service name property defined as an expression, in which case the expression will be evaluated dynamically for each invocation (for example, for each message which will be relayed) prior to determining the attachment point.

The RVI Relay service is added to the iSM channel by implementing and assigning a corresponding process flow containing the service.

The RVI Relay service is synchronous. Depending on the timeout settings, this service will wait for a response document from the gateway before proceeding. The response document will include content, a header, and user special registers (SREGs). To return a SREG from the gateway, the SREG must be in *message* scope as local and flow scopes are cleared when the process flow running on the gateway ends.

Note: This section describes how to configure an RVI Relay service. To construct a fully populated iSM channel, incorporate the service into a process and then include the process as a route of the channel. For more information on how to design and build a channel, see the *iWay Service Manager User's Guide*.

To configure the RVI Relay service, you must create a process flow for your application project using iWay Integration Tools (iIT) and select *RVI Relay: send message to gateway service (com.ibi.agents.RVIRelay)* as the agent type for the Server Agent component. For more information, see *Configuring a Service Using iWay Integration Tools* on page 87.

For a complete description of the configuration parameters that are available for the RVI Relay service, see *RVI Relay Service Configuration Parameters* on page 63.

Reference: RVI Relay Service Configuration Parameters

The following table lists and describes parameters for the RVI Relay service.

Property	Description
Service Name (required)	Name of the service that is supported by an Executor Server attach point. Service names should be short and descriptive. Service names are case-sensitive and may not contain punctuation or other special characters. This service name must be identical to the service name that is specified during the configuration of the gateway listener, since it refers to the service offered by the gateway.
Tolerance	Period to wait for an Execution server offering the correct service to be available. The default value is 30 seconds.
Timeout	Maximum time period to wait for a response from the executing service. The default value is 30 seconds.
Attempt Retry	If set to true, failed connections to the execution server will be retried. The default value is true.
Output On Failure	If the relay operation is unsuccessful, this determines whether the agent returns the standard error document or its input. If input is selected, the error document will be stored in the rvi.status register. The default value is error.

Property	Description
Method of compression to use (required)	The form of compression that should be used on the output: ❑ none ❑ smallest ❑ fastest ❑ standard ❑ Huffman The default value is none.

Configuring the RVIGateway Listener

The RVIGateway listener offers one service to one attach point. Each active RVIGateway listener offers service attachments to one attach point on a receiving engine. One channel is offered for each possible simultaneous execution. This is configured as the thread count for the listener. The number of offered channels will not grow by demand, although the gateway will attempt to reinstate a failing channel.

To configure the RVIGateway listener, you must create a channel for your application project using the Channel Builder in iWay Integration Tools (iIT) and select *RVIGateway* as the listener type for your inlet. For more information, see *Configuring a Listener Using iWay Integration Tools* on page 81.

For a complete description of the configuration parameters that are available for the RVIGateway listener, see *RVIGateway Listener Configuration Parameters* on page 64.

Reference: RVIGateway Listener Configuration Parameters

The following table lists and describes parameters for the RVIGateway listener.

Note: Parameters that are common to most listeners are described in *Listener Configuration Parameters* on page 95.

Property	Definition
Attach Point Host (required)	Host address of the attach point, which can be a list such as: `host1:1234;host2:3456(ipi bind address)` The list can also be stored as a file using the iFL _file() function.
Attach Point Port (required)	Socket port where the attach point is listening for gateway connections. This will be the default port, used if a host does not carry the port as host:port. **Note:** The value for the Attach Point Port parameter must not be zero (0) or blank.
SSL Context Provider	Defined iWay Security Provider for SSL Context.
Service Name (required)	Name of the service that is supported by an Executor Server attach point. The service name is a locator that identifies the channel or listener that runs on the specified machine name. Therefore, it represents a combination of the channel name and the machine and port name for remote invocation. In addition, this is the service name that is referred to in the relay service at the attach point.
Reverify time	Period of time (in seconds) to verify the presence of the attach point. The default value is 120 seconds.
Read Timeout	Period, in seconds, to wait for a response from the attach point. The default value is 1.0 seconds.
Preserve Stream	If set to *true*, an incoming RVI stream message will be processed as a stream document containing the input stream for the message. The default value is false.
IP Interface Host	Local IP interface from which the outgoing IP socket originates. This field is usually left blank.

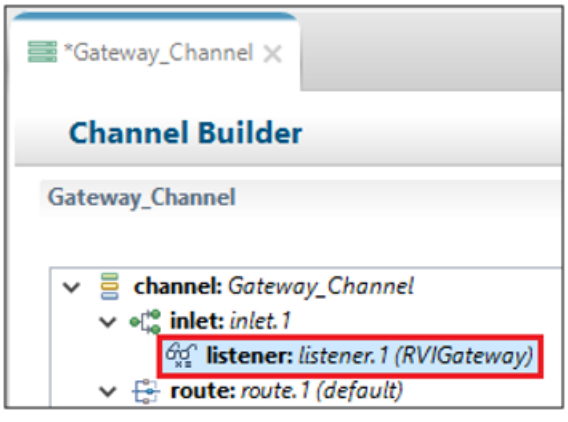
Configuring a Service to Test the Reverse Invocation

The gateway listener performs the action requested by the relay service. For example, if a database operation is required to be performed, but the service is available on the gateway machine, the gateway listener picks up the message from the relay service and completes the processing. The result is then returned to the relay service or relay channel that is configured.

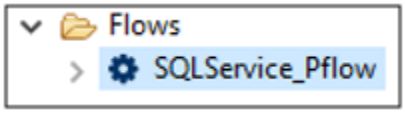
Procedure: How to Create a Service on the Gateway

To create a service on the gateway:

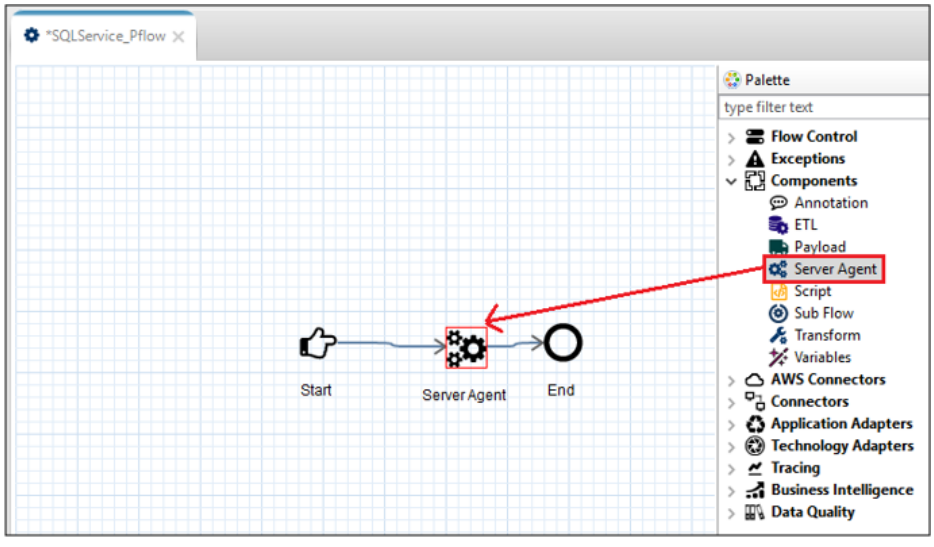
1. Configure a new channel (for example, *Gateway_Channel*) using iWay Integration Tools (iIT).
2. Configure a RVIGateway listener as an inlet for this channel, as shown in the following image. For more information, see *Configuring the RVIGateway Listener* on page 64.



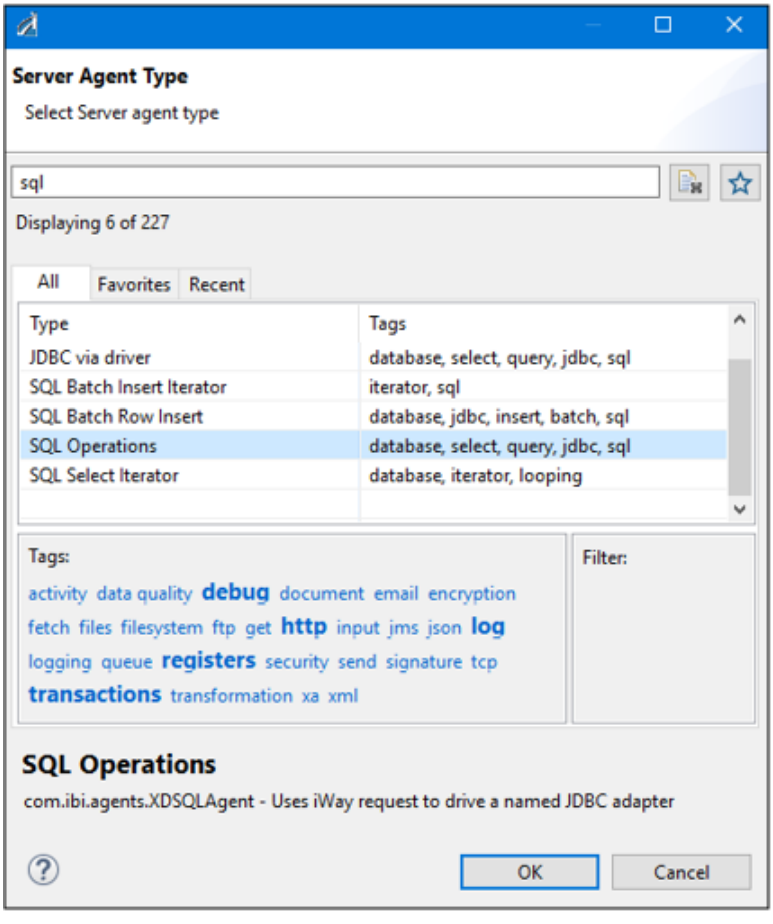
3. Create a new process flow (for example, *SQLService_Pflow*), as shown in the following image.



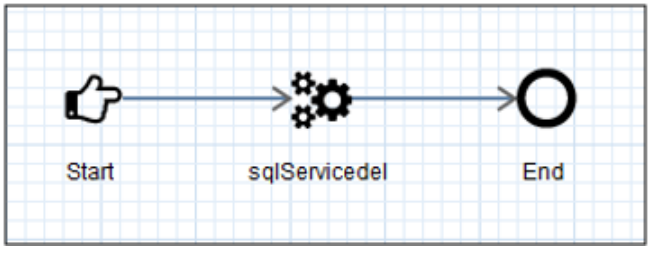
4. Add the *Server Agent* component from the Palette to the process flow, as shown in the following image.


*SQLService_Pflow
Palette
type filter text
Flow Control
Exceptions
Components
Annotation
ETL
Payload
Server Agent
Script
Sub Flow
Transform
Variables
AWS Connectors
Connectors
Application Adapters
Technology Adapters
Tracing
Business Intelligence
Data Quality
Start
Server Agent
End

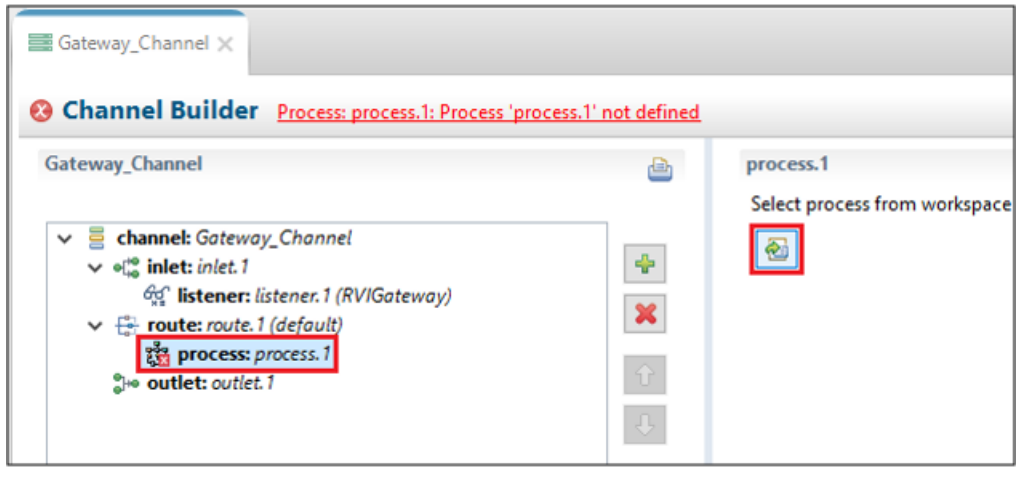
5. Select *SQL Operations (com.ibi.agents.XDSQLAgent)* as the service type, as shown in the following image.



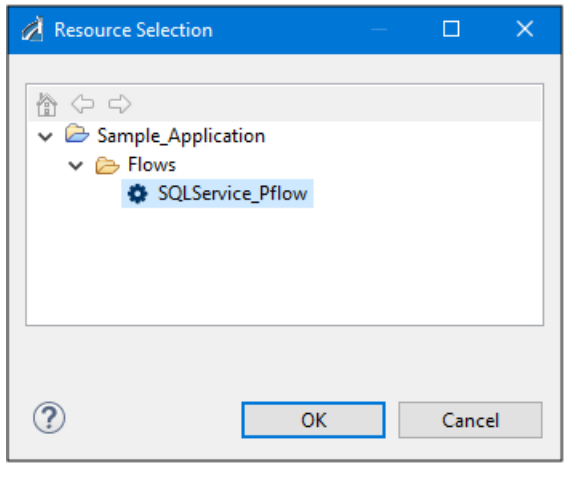
This service would be invoked by the relay service on the proxy machine through the socket call. In this case, the SQL object (sqlServicedel) is used to perform a database operation (for example, a delete action), as shown in the following image.



6. Open the channel that you configured earlier (for example, *Gateway_Channel*).
7. In the Channel Builder, select *process: process.1* under the route node in the left pane and then click the *Resource Selection* icon in the right pane, as shown in the following image.

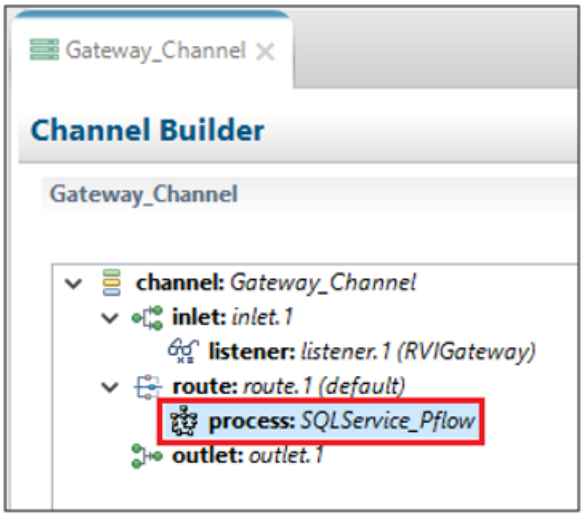


The Resource Selection dialog box opens, as shown in the following image.

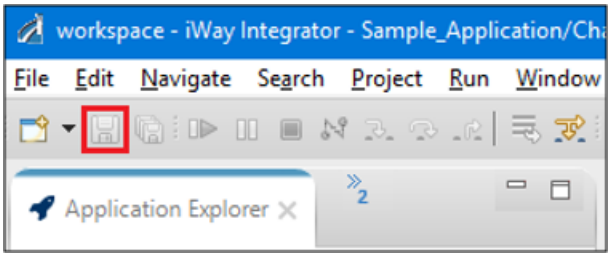


8. Expand the *Flows* subfolder, select the process flow you that configured earlier (for example, *SQLService_Pflow*), and then click *OK*.

The process flow is now associated as the route of your channel, as shown in the following image.



9. Click *Save* to save all of the changes you made to your channel, as shown in the following image.

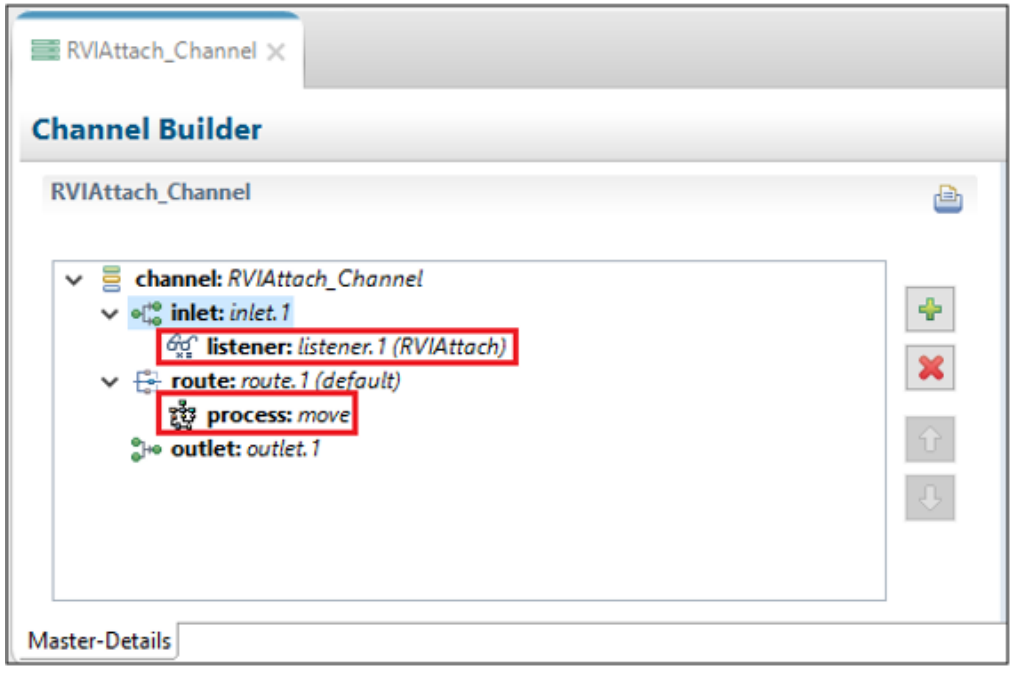


Procedure: How to Configure the RVIAttach Channel

To configure the RVIAttach channel:

1. Configure a new channel (for example, RVIAttach_Channel) in an application project using iWay Integration Tools (iIT).
2. Configure an RVIAttach listener as an inlet for this channel. For more information, see *Configuring the RVIAttach Listener* on page 61.

3. Configure this channel (RVIAttach_Channel) to perform the initial handshake with the gateway channel, as shown in the following image. Note that the move route simply contains a start-end process flow which passes the message along.



4. Save the channel (RVIAttach_Channel).

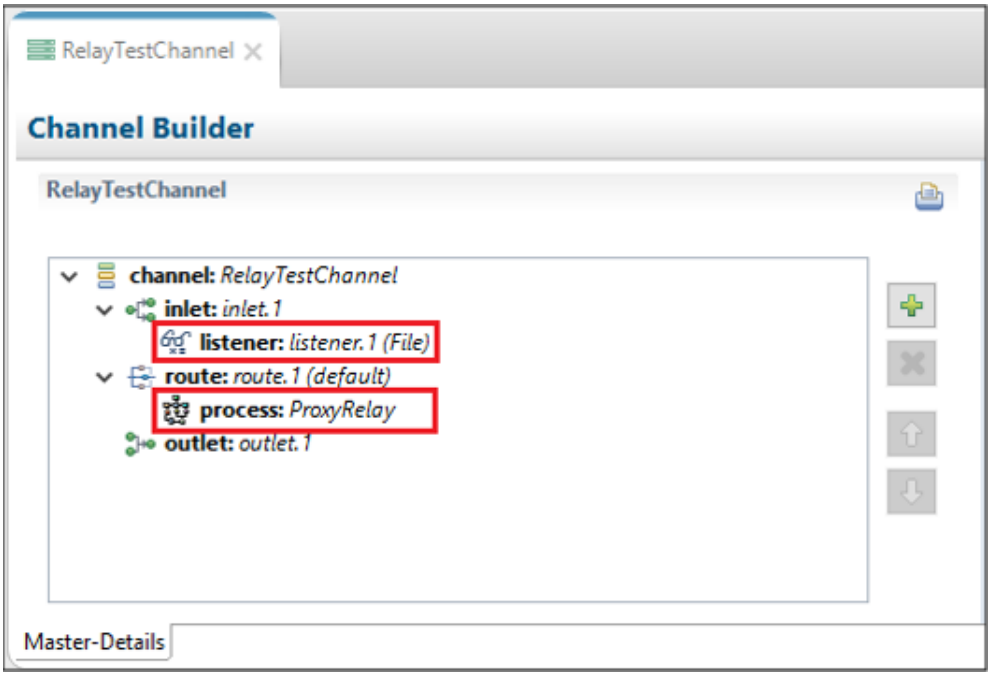
Procedure: How to Configure the Channel to Invoke the Remote Gateway Service

As an example, assume that a channel exists with a File listener that picks up files from a specified directory. After the file is picked up, a service on the gateway is invoked through the attach point and the result is written to an output directory.

To configure the channel to invoke the remote gateway service:

1. Configure a new channel (for example, RelayTestChannel) in an application project using iWay Integration Tools (iIT).
2. Configure a File listener as an inlet for this channel..
3. Configure a new process flow (for example, ProxyRelay) which includes the RVI Relay service (com.ibi.agents.RVIRelay). Add this process flow as the route of your channel.

4. Configure this channel (RelayTestChannel) to test the remote service on the gateway machine, as shown in the following image.



5. Save the channel (RelayTestChannel).

Procedure: How to Test the RVI Invocation Using the Attach Point and Gateway

To test the RVI invocation using the attach point and gateway:

1. Deploy the application project that contains the *RVIAttach_Channel* you configured in *Configure the RVIAttach Channel*.
2. If this application project deploys successfully (without any errors), then deploy the application project that contains the *Gateway_Channel* on the gateway machine. This is the channel you configured in *How to Create a Service on the Gateway* on page 66.

   If this application project deploys successfully (without any errors), then a successful connection between the attach point and the gateway has been established.
3. Deploy the application project that contains the *RelayTestChannel* to invoke the RVI Relay service. This is the channel you configured in *How to Configure the Channel to Invoke the Remote Gateway Service* on page 71.
4. Place a file in the input directory that was configured for the File listener (Input Path parameter) in the *RelayTestChannel* to start the invocation process.

The file read is successful indicating a success test run on the RVIAttach side. To see if the gateway service was invoked successfully, check the database to see if the database operation was completed successfully on the gateway side. If the database operation was completed, then this indicates that the gateway service ran successfully.

Chapter 5

Asynchronous Forward Transfer Invocation Queue Processing

This section describes how to configure Asynchronous Forward Transfer Invocation (AFTI) queue processing.

In this chapter:

Asynchronous Forward Transfer Invocation Overview

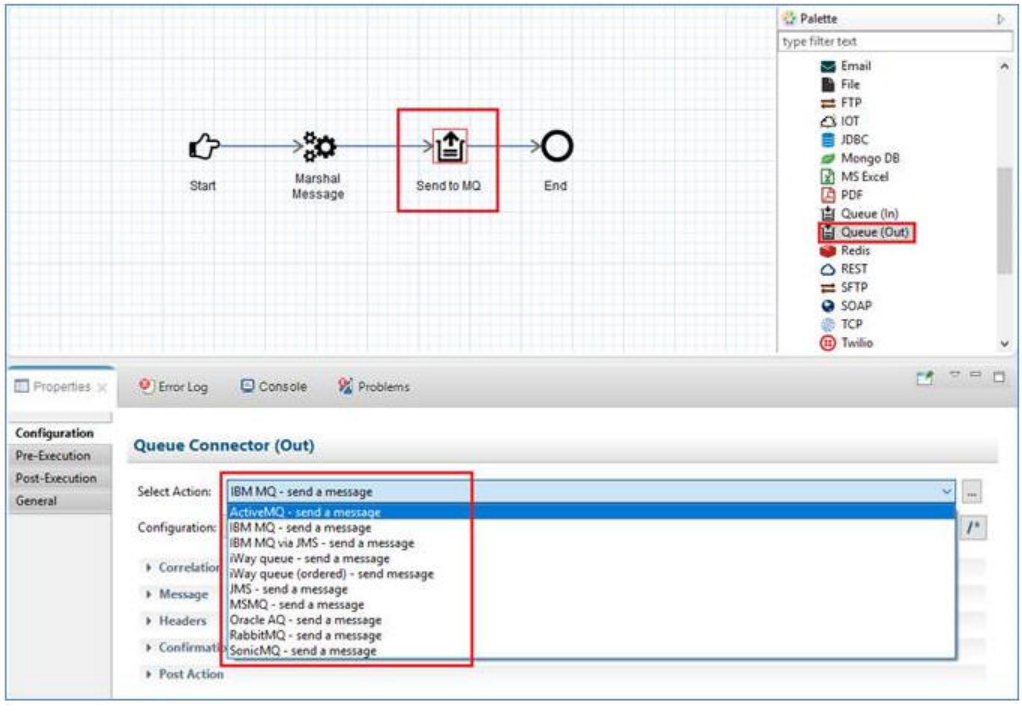
The cross-channel protocols that are described in the previous chapters of this documentation are completely managed by iSM. Sometimes an application requires a cross-channel structure that must use another protocol, such as MQ Series, MSMQ, FTP, or any other protocol that is not specific to iSM. To support this requirement, it is necessary to marshall the message and its context for transmission into a format that iSM can support. Asynchronous Forward Transfer Invocation (AFTI) allows a service to transfer a message and its context across a channel using a protocol that is not specific to iSM. There is no restriction on the type of protocol that can be used.

In iSM terms, marshalling refers to the serialization of the current message and its context (special registers) for saving or transmission purposes. Unmarshalling takes a marshalled serialized message and restores its content and context.

AFTI is accomplished through the use of a provided service within a process flow. To send a context and a message to another party, configure the Marshall preemitter (com.ibi.preemit.XDMarshall). When the message is received by an iSM channel, it is automatically unmarshalled. The message context is reestablished and the message flows through the standard channel functionality.

Additionally, the Marshall service (com.ibi.agents.XDMarshallAgent) is available, which can be positioned to marshall the message and its context before it is passed to an emit service.

The following image shows that the Queue (Out) object was added to the process flow, renamed as *Send to MQ*, and that a message queue type is being specified from the Select Action drop-down list.



On the receiving side, a simple process flow can deposit the unmarshalled message into the desired internal queue. A best practice can be to put the name of the desired queue in a Special Register (SREG) if the process flow can deposit into different queues.

AFTI offers optional data compression and encryption through AES. Encryption is offered, since the marshalled message may reside on an intermediate media (for example, a file system). Any message-bearing protocol can be used. If the marshalled message will never be serialized to external media, then encryption is probably not required. While decompression is handled automatically by the receiving channel, the encryption key must be supplied to the receiver. This is the purpose of the AES Key parameter, which is available for all iSM listeners.

Configuring a Marshalls a Message Service

To configure a Marshalls a message service, you must create a process flow for your application project using iWay Integration Tools (iIT) and select *Marshalls a message (com.ibi.agents.XDMarshallAgent)* as the agent type for the Server Agent component. For more information, see *Configuring a Service Using iWay Integration Tools* on page 87.

For a complete description of the configuration parameters that are available for the Marshall service, see *Marshall Service Parameters* on page 77.

For a complete description of the edges that are returned by the Marshall service, see *Marshalls a Message Service Edges* on page 78.

Reference: Marshall Service Parameters

The following table lists and describes parameters for the Marshalls a message service.

Parameter	Description
Compress Messages (required)	Supports optional compression. The default value is true.
Marshall User Special Registers	Set to *true* if the marshalling is to include user registers. Normally the marshaller passes header, document, and system registers associated with the message. The default value is false.
Namespace	If empty or an asterisk character (*) is entered, then all namespaces are marshalled. If a namespace is entered, then only registers in the specified namespace are marshalled.
Use Encryption (required)	Set to *true* if the marshalled messages should be encrypted after compression. The AES Key parameter must also be configured if this parameter is set. The default value is false.

Parameter	Description
AES Key	Must be the same value on both sides. Maximum length is 16 characters, but can include escapes to allow the use of binary values. For more information, see the _aes() iFL function in the *iWay Functional Language Reference Guide*. This key can be generated by an iFL statement, such as storing the key in a special register accessed at runtime. Once set, the key cannot be altered. Use of AES encryption can be slow, and should only be used when the marshalled message can appear in a publicly accessible area.

Reference: Marshalls a Message Service Edges

The following table lists and describes the edges that are returned by the Marshalls a message service.

Edge	Description
OnSuccess	Operation was successful.
OnFailure	Fail condition occurred during execution.
OnError	Exception condition occurred during execution.
OnFailedOperation	Could not perform the operation requested.

Configuring an Unmarshalls a Message Service

To configure an Unmarshalls a message service, you must create a process flow for your application project using iWay Integration Tools (iIT) and select *Unmarshalls a message (com.ibi.agents.XDUnmarshallAgent)* as the agent type for the Server Agent component. For more information, see *Configuring a Service Using iWay Integration Tools* on page 87.

For a complete description of the configuration parameters that are available for the Unmarshall service, see *Unmarshalls a Message Service Parameters* on page 79.

For a complete description of the edges that are returned by the Unmarshall service, see *Unmarshalls a Message Service Edges* on page 79.

Reference: Unmarshalls a Message Service Parameters

The following table lists and describes parameters for the Unmarshalls a message service.

Parameter	Description
Use encryption (required)	Set to *true* if the unmarshalled messages should be encrypted after compression. The AES Key parameter must also be configured if this parameter is set. The default value is false.
AES Key	Must be the same value on both sides. Maximum length is 16 characters, but can include escapes to allow the use of binary values. For more information, see the _aes() iFL function in the *iWay Functional Language Reference Guide*. This key can be generated by an iFL statement, such as storing the key in a special register accessed at runtime. Once set, the key cannot be altered. Use of AES encryption can be slow, and should only be used when the unmarshalled message can appear in a publicly accessible area.

Reference: Unmarshalls a Message Service Edges

The following table lists and describes the edges that are returned by the Unmarshalls a message service.

Edge	Description
OnSuccess	Operation was successful.
OnFailure	Fail condition occurred during execution.
OnError	Exception condition occurred during execution.
OnFailedOperation	Could not perform the operation requested.

Chapter 6

Configuring iWay Service Manager Components

During the cross-channel services configuration process, you are required to configure listeners and services using iWay Integration Tools (iIT). This section provides the steps that are needed to access and configure these iSM components. Descriptions of the parameters for each component is provided within the corresponding sections.

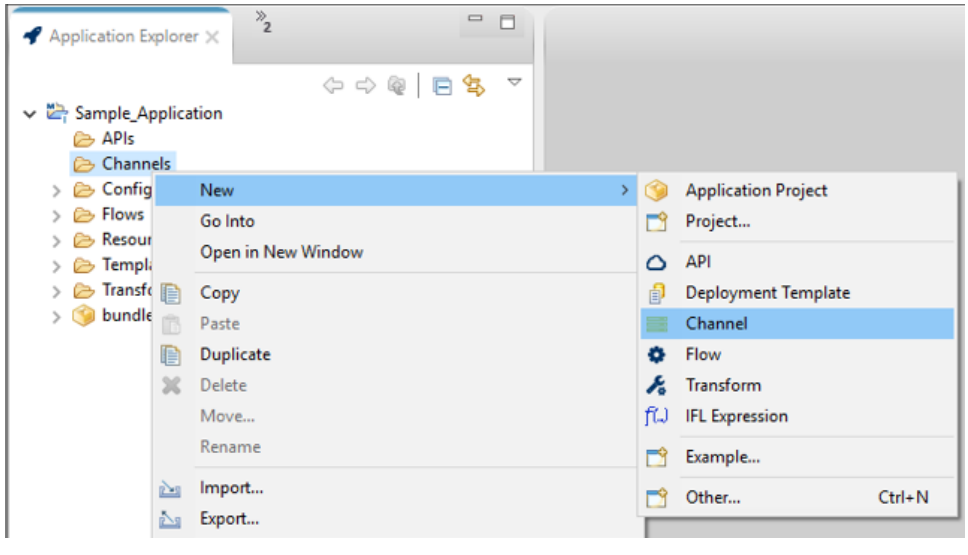
In this chapter:

Configuring a Listener Using iWay Integration Tools

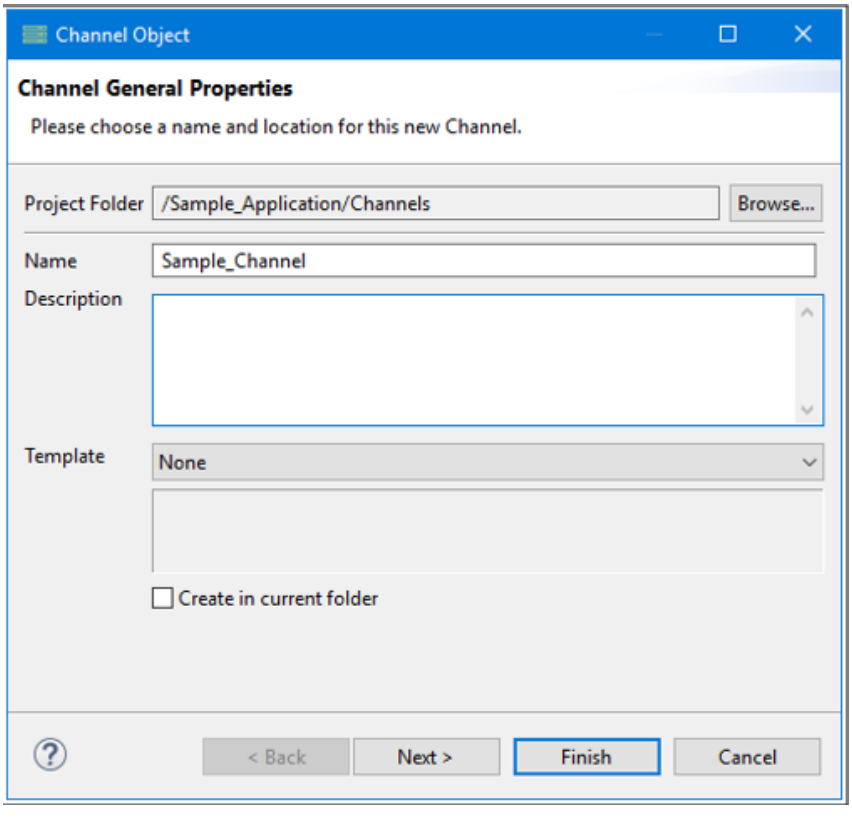
This section describes how to configure a listener using iWay Integration Tools (iIT).

Procedure: How to Configure a Listener Using iWay Integration Tools

1. Using iWay Integration Tools, create a new channel within an application project. Right-click the *Channels* subfolder under your application project, select *New*, and then click *Channel* from the context menu, as shown in the following image.

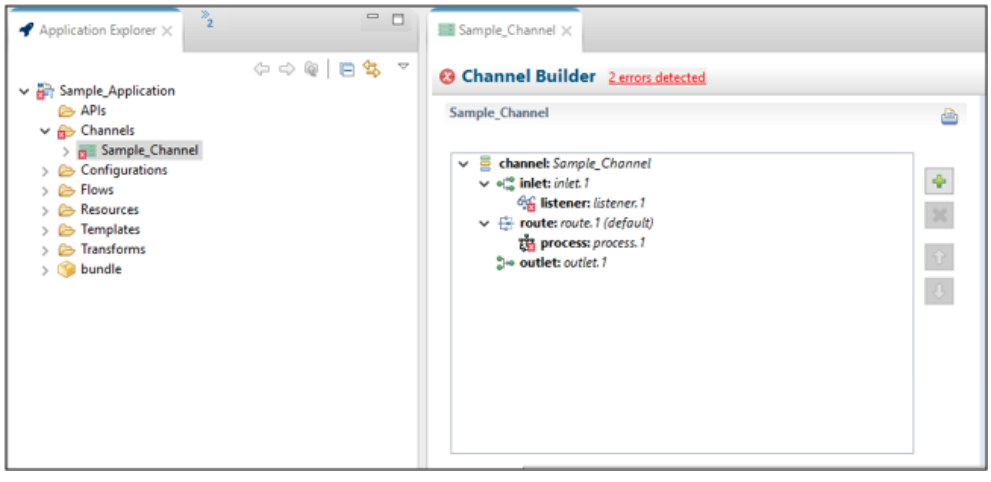


The Channel Object dialog box opens, as shown in the following image.

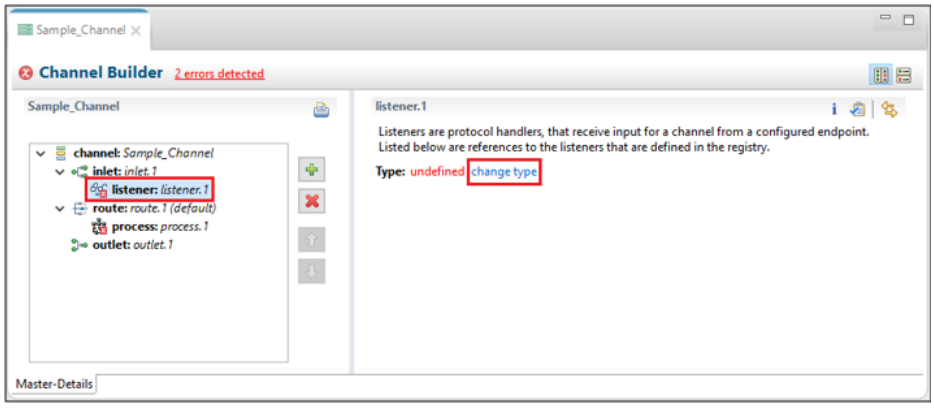


2. Specify a name (required) and a brief description (optional) for your channel and then click *Finish*.

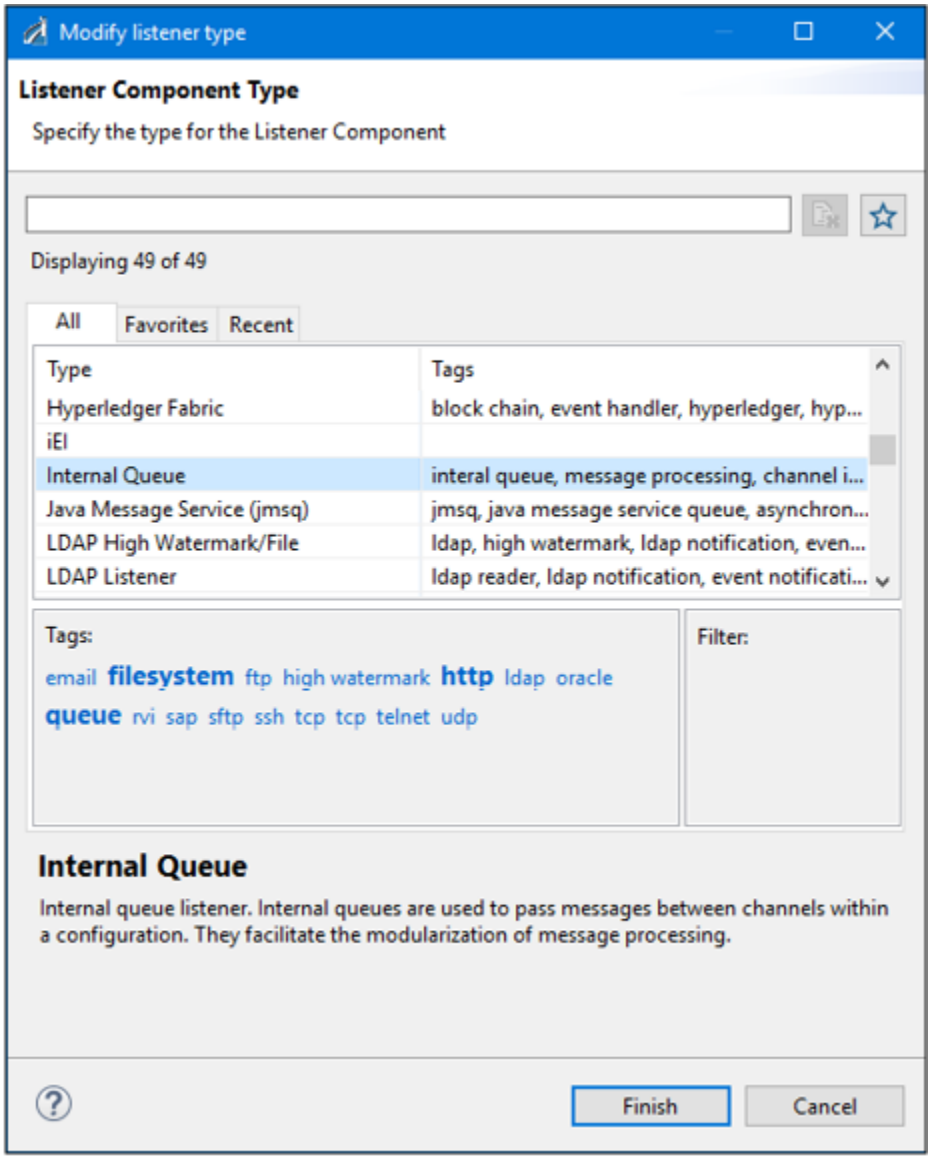
The new channel appears as a node under your Channels subfolder in the left pane. The Channel Builder also opens as a new tab in the workspace area. The name of this tab corresponds to the channel name you specified (for example, *Sample_Channel)*, as shown in the following image.



3. In the left pane of the Channel Builder, click *listener: listener.1* under the inlet:inlet.1 node, and then click *change type*, as shown in the following image.

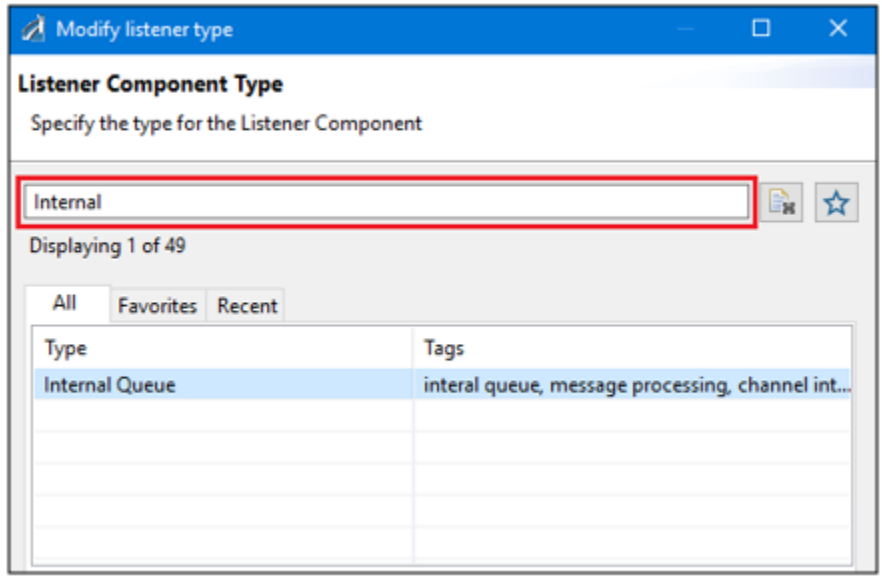


The Modify listener type dialog box opens, as shown in the following image.



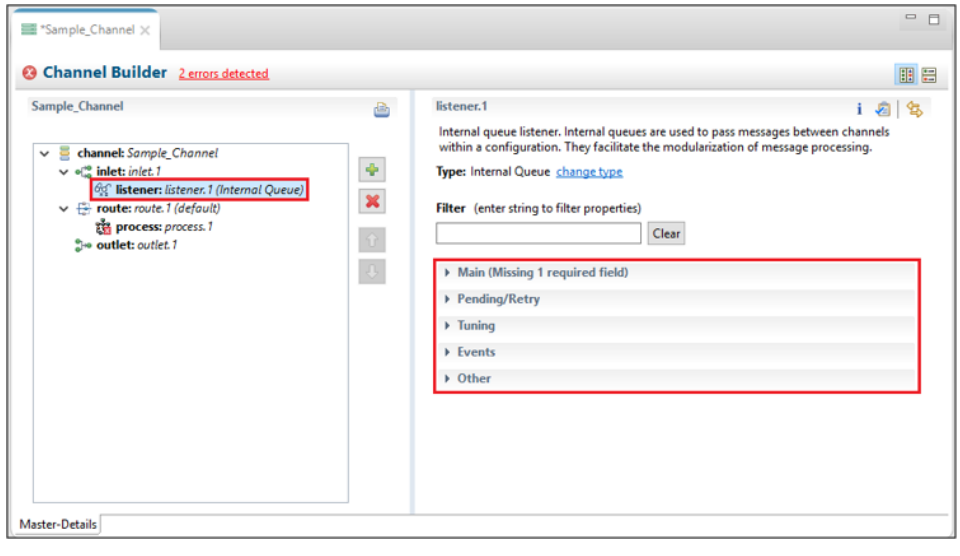
4. Scroll through the list of available listeners and select the specific listener that you want to configure (for example, *Internal Queue*).

Note: You can also quickly filter through the list by typing part of the listener name in the filter field, as shown in the following image.



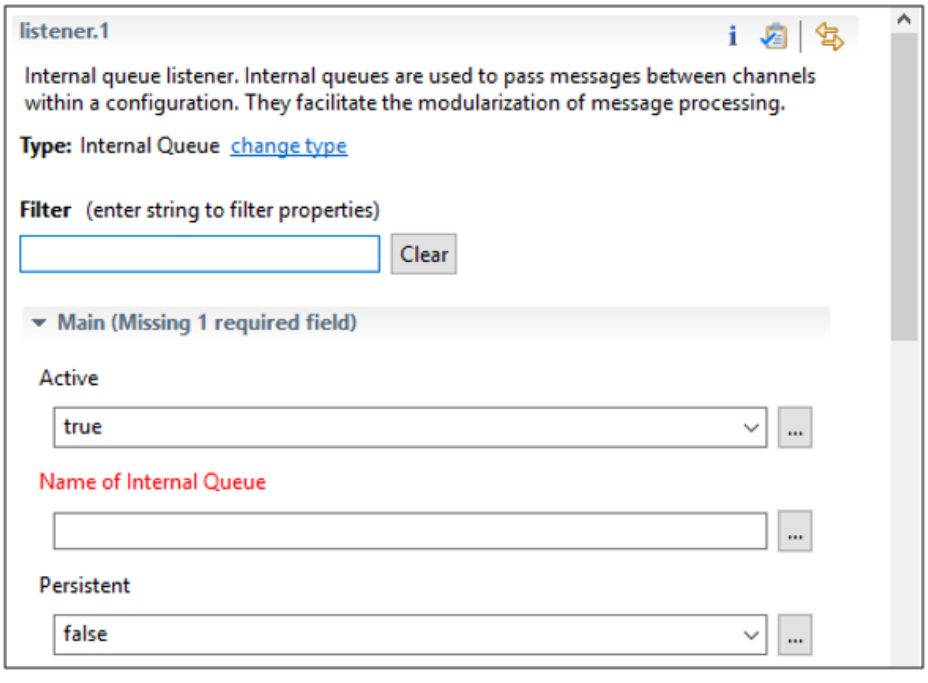
5. After you have selected your listener in the Modify listener type dialog box, click *Finish*.

   The Channel Builder is refreshed with your selected listener, as shown in the following image.

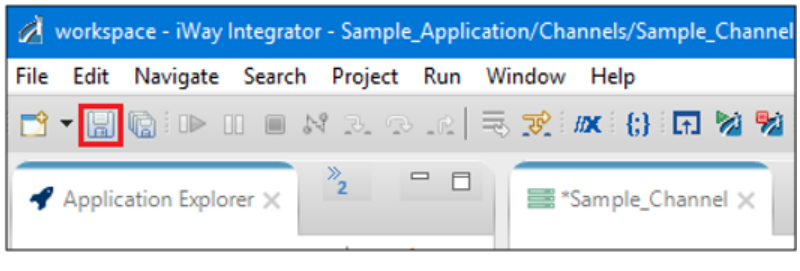


Notice that the name of the listener is appended to the *listener: listener.1* node in the left pane. Configuration parameters for the selected listener are organized into expandable groups, which can be accessed in the right pane.

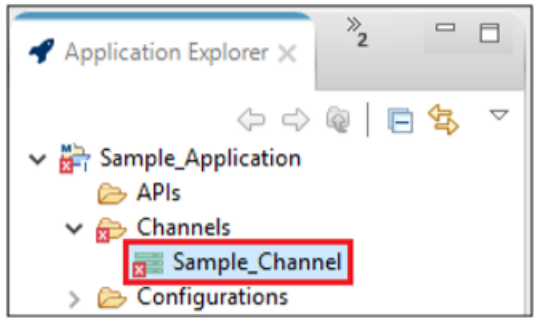
In the following image, the Main configuration parameter group has been expanded. The name of any required parameter appears in red.



6. Provide the appropriate values for the configuration parameters as required for the selected listener.
7. Click *Save* to save any changes you made to your listener and/or channel, as shown in the following image.



To modify your listener and/or channel at any point, double-click the channel under your application project, as shown in the following image.



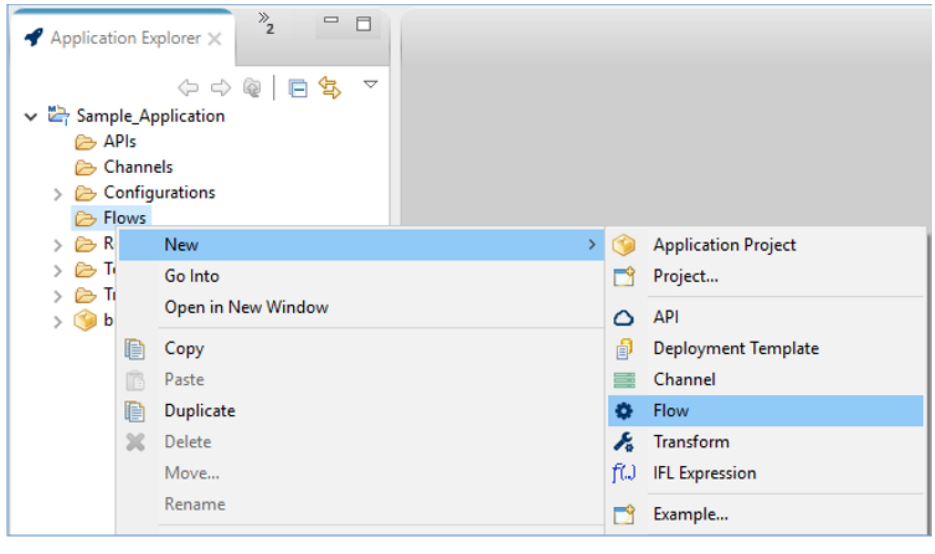
The Channel Builder will open as a tab in the workspace area.

Configuring a Service Using iWay Integration Tools

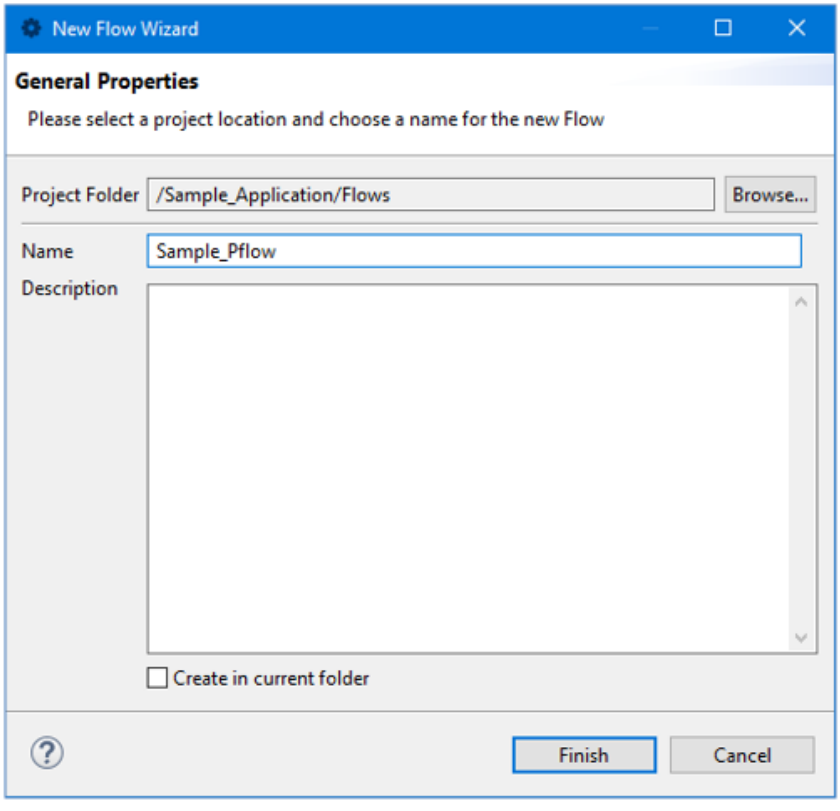
This section describes how to configure a service using iWay Integration Tools (iIT).

Procedure: How to Configure a Service Using iWay Integration Tools

1. Using iWay Integration Tools, create a new process flow within an application project. Right-click the *Flows* subfolder under your application project, select *New*, and then click *Flow* from the context menu, as shown in the following image.

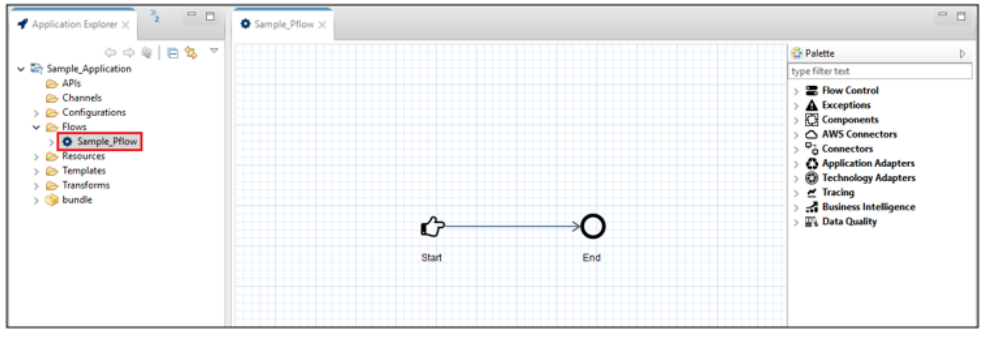


The New Flow Wizard opens, as shown in the following image.



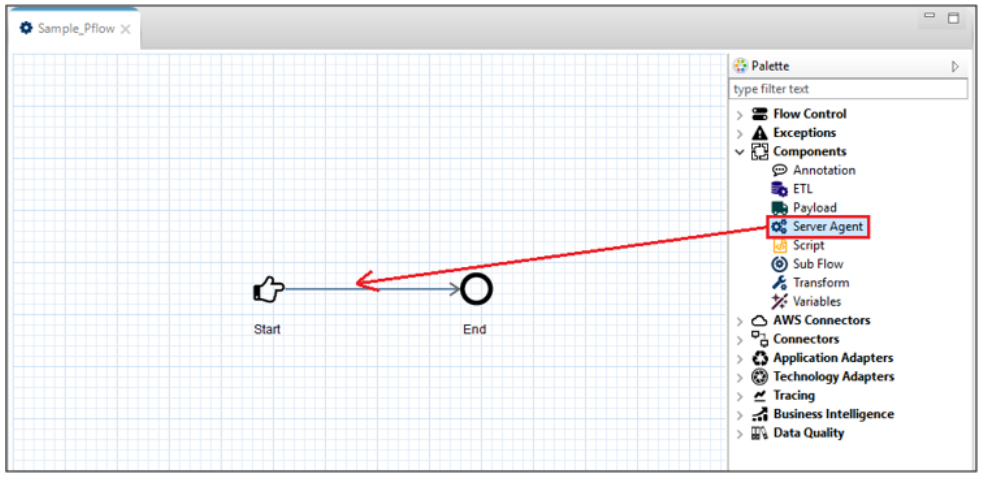
2. Specify a name (required) and a brief description (optional) for your process flow and then click *Finish*.

The process flow opens as a new tab in the workspace area. The name of this tab corresponds to the process flow name you specified (for example, *Sample_Pflow)*, as shown in the following image.

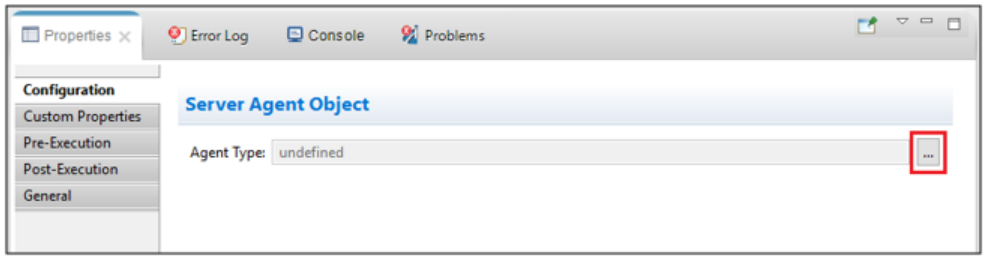


The new process flow also appears as a node under your Flows subfolder in the left pane.

3. From the Palette located on the right pane, expand the *Components* category.
4. Click and drag the *Server Agent* component to the workspace area between the Start and End objects, as shown in the following image.

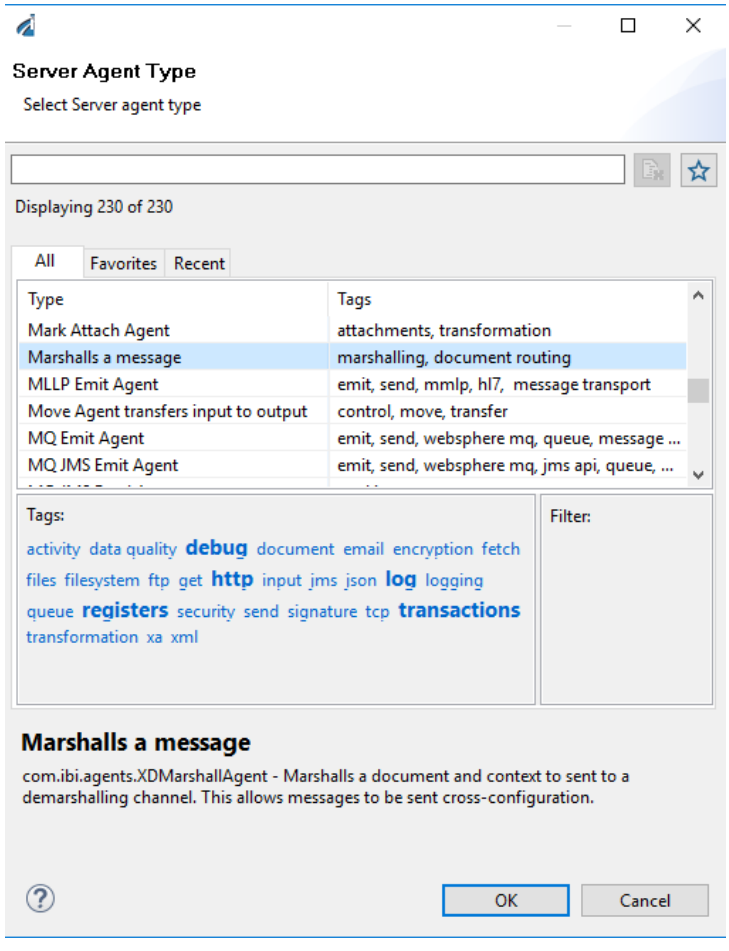


The Server Agent Object Properties pane opens as a tab below the workspace area, as shown in the following image.



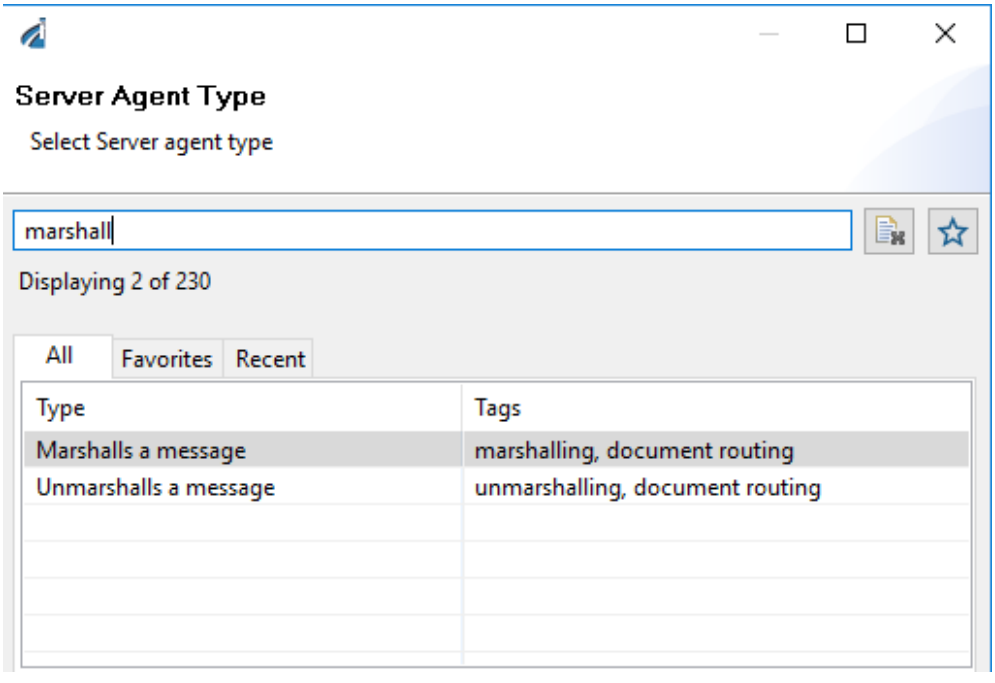
5. Click the ellipses button next to the Agent Type field.

The Server Agent Type dialog box opens, as shown in the following image.



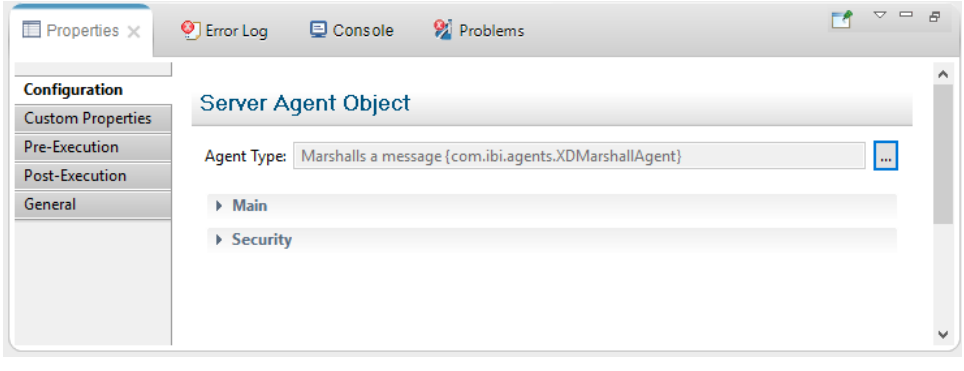
6. Scroll through the list of available services (agents) and select the specific service that you want to configure. For example, *Marshalls a message (com.ibi.agents.XDMarshallAgent)*.

Note: You can also quickly filter through the list by typing part of the service name in the filter field, as shown in the following image.



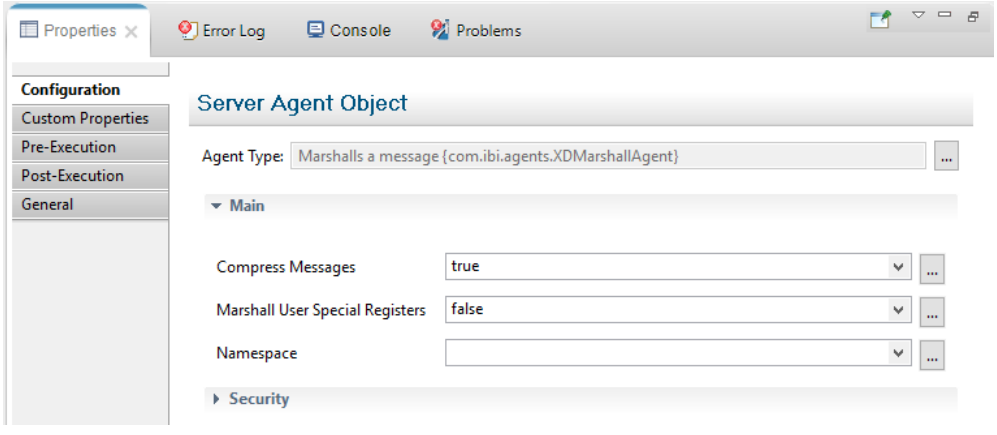
7. After you have selected your service in the Server Agent Type dialog box, click *OK*.

   The Server Agent Object Properties pane is refreshed with your selected service, as shown in the following image.

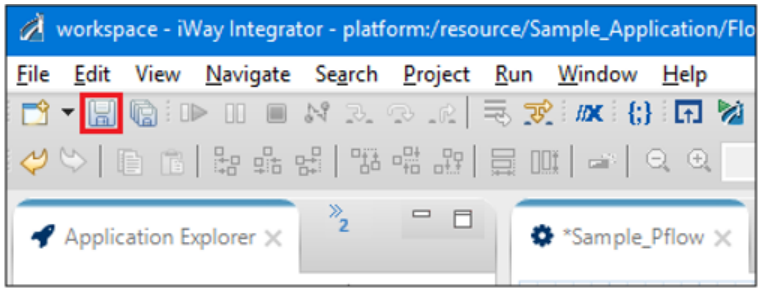


Configuration parameters for the selected service are organized into expandable groups.

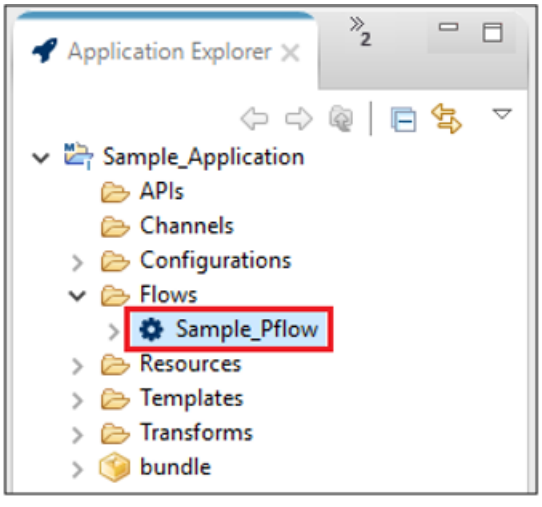
In the following image, the Main configuration parameter group has been expanded. The name of any required parameter appears in red.



8. Provide the appropriate values for the configuration parameters as required for the selected service.
9. Click *Save* to save any changes you made to your service and/or process flow, as shown in the following image.



To modify your process flow at any point, double-click the process flow under the Flows subfolder of your application project, as shown in the following image.



The process flow will open as a tab in the workspace area.

Chapter 7

Common Configuration Parameters

This section provides a reference for common configuration parameters used by iWay Service Manager (iSM) components (for example, listeners and services).

In this chapter:

Listener Configuration Parameters

The following table lists and describes common parameters used by the Internal, Ordered, and RVI Queue listeners.

Tuning Parameters

Parameter	Description
Multithreading	Indicates the number of worker threads (documents or requests) that iWay Service Manager can handle in parallel. Setting this to a value of greater than 1 enables the listener to handle a second request while an earlier request is still being processed. The total throughput of a system can be affected by the number of threads operating. Increasing the number of parallel operations may not necessarily improve throughput. The default value is 1. The maximum value is 99.
Maximum Threads	The parallel threads can grow to this count automatically on demand. Over time, the worker count will decrease back to the multithreading level. Use this parameter to respond to bursts of activity. The default value is 1.

Parameter	Description
Optimize Favoring	Use this option to customize how the listener performs. For smaller transactions, select *performance*. For large input documents that could monopolize the amount of memory used by iWay Service Manager, select *memory*. The default value is performance.
Polling Interval	The maximum wait interval (in seconds) between checks for new requests or commands. The higher this value, the longer the interval, and the fewer system resources that are used. The side effect of a high value is that the worker thread will not be able to respond to a stop command. The default value is 2.0 seconds.

Events Parameters

Parameter	Description
Expired Retry Flow	Name of a published process flow to run if a message on the retry queue has expired.
Failed ReplyTo Flow	Name of a published process flow to run if a message cannot be emitted on an address in its reply address list.
Dead Letter Flow	Name of a published process flow to run if an error cannot be emitted on an address in its error address list.
Channel Failure Flow	Name of a published process flow to run if this channel cannot start or fails during message handling. iWay Service Manager will attempt to call this process flow during channel shut down due to the error.
Parse Failure Flow	Name of a published process flow to run if XML or JSON parsing fails for the incoming message.
Channel Startup Flow	Name of a published process flow to run prior to starting the channel.

Parameter	Description
Channel Shutdown Flow	Name of a published process flow to run when the channel is shut down.
Startup Dependencies	A comma-separated list of channel names that must be started before this one is called.

Other Parameters

Parameter	Description
Whitespace Normalization	Specifies how the parser treats whitespace in Element content. Choose *preserve* to turn off all normalization as prescribed by the XML Specification. Choose *trim* to remove extra whitespace in pretty printed documents and for compatibility with earlier versions. The default value is preserve.
Input Format	If set to *true*, the input data is sent directly to the business logic step. The data is not preparsed, parsed, or validated. This flag is used primarily to send non-XML to the business logic or replyTo without processing it. Select one of the following options from the drop-down list: ❏ No parse, input is flat {flat} ❏ Parse as JSON {json} ❏ Parse as XML {xml) The default value is XML.
Execution Time Limit	The maximum time that a request may take to complete. Used to prevent runaway requests. Any request that takes longer to complete than this value will be attempted to be terminated.
Default Java File Encoding	The default encoding if the incoming message is not self-declaring (that is, XML).

Parameter	Description
Agent Precedence	Sets the order by which iWay Service Manager selects agents. iWay Service Manager selects the agent or agents to process the document by searching through the configuration dictionary. Usually, it looks for a document entry in the configuration and when a match is found, the agent specified in that document entry is selected. If a matching document entry is not found, or no agent is specified, the engine looks in the input protocol configuration (listener). To have the processing agent taken directly from the listener (thus ignoring the document entry), use <listener> overrides <document>. Possible values are <document> overrides <listener> and <listener> overrides <document>. The default value is <document> overrides <listener> {1}.
Always reply to listener default	If set to *true*, the default reply definition is used in addition to defined reply-to and error-to destinations. The default value is false.
Error Documents treated normally	If set to *true*, error documents are processed by any configured preemitters. The default value is false.
Listener is Transaction Manager	If set to *true*, agents run within a local transaction. The default value is false.
Record in Activity Log(s)	If set to *true*, activity on this channel will be recorded in the activity logs, otherwise the activity will not be recorded. The default value is true.
AES Key	If the channel will receive encrypted AFTI messages, set the AES key (maximum 16 characters) to be used for decrypting.
Startup Dependencies	A comma-separated list of channel names that must be started before this one is called.

Service Configuration Parameters

The following table lists and describes common parameters used by the Internal and Ordered emit services.

Parameter	Description
Avoid Preemitter	Determines whether any preemitter should be avoided. Select one of the following options from the drop-down list: ❑ true (default) ❑ false The default value is true.
Respect Transactionality	Determines whether this emit service should post messages regardless of the commit/rollback state of the transaction. For example, you may not want to respect transactionality when passing messages that reflect the progress of an application or errors within the application. The default value is true.
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.

Chapter 8

Deploying iWay in a High Availability Environment

The following section describes how to deploy iWay Service Manager in a high availability environment and manage server failover.

In this chapter:

High Availability Overview

High Availability (HA) describes the ability of a system to accept and process transactions a great percent of the time, achieving as close to 100% as technically possible. The features and characteristics of a specific software product are not solely responsible for the ability of a system to be *highly available*. For example, choosing high reliability hardware, and ensuring uninterruptible power, network connectivity, and sufficient capacity and throughput are all essential to achieving high availability.

There are specific architectural mechanisms and design patterns employed to make a system highly available, the most important being failover and scaling. iWay is compatible with architectures comprising third-party HA solutions and also has its own native features to facilitate HA.

Failover

Failover is the capability to switch over automatically to a redundant or *hot* standby host or subsystem upon the failure or abnormal termination of the primary host or subsystem. Ideally, failover is accomplished without manual intervention since failures generally occur without warning.

Scaling and Load Balancing

Vertical Scaling refers to increasing the processing capability of a host system. This is accomplished by adding processors, memory, faster storage, and so on. Vertical scaling is primarily a hardware effort that does not affect the system topology or software configuration.

Horizontal Scaling refers to increasing the number of hardware systems hosting the software. For example, two hardware hosts running iWay achieve roughly double the throughput of one, assuming other dependent resources are available and adequately performing. Effectively distributing the workload across two or more iWay Service Manager instances is referred to as *load balancing* and is a key factor in achieving maximum throughput with horizontal scaling. Supporting adequate throughput is an important aspect of HA, because while a system may be online, if it is running at or close to capacity, it may appear unavailable to clients. Scaling directly addresses the throughput issue, and also provides some of the benefits of failover because it eliminates a single point of failure. Failure of one host (out of two or more) will not make the supported service(s) unavailable, although it may impact throughput and response times until the failed host has *failed over* to its backup or is brought back online.

Implementing High Availability

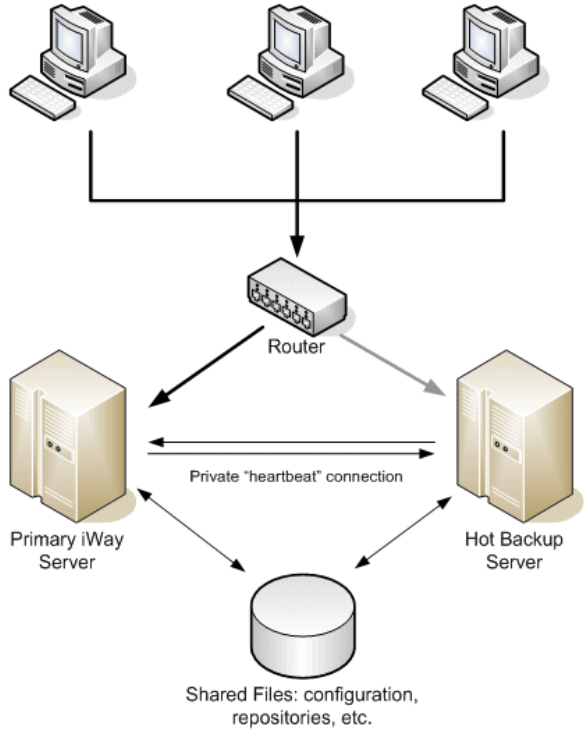
There are a number of strategies and techniques you can use to implement high availability in your environment, including:

- Simple failover using iWay heartbeat
- Simple failover using third-party tools
- IP-based horizontal scaling
- Web-based horizontal scaling
- Web-based scaling using iWay Performance Monitor
- Horizontal scaling for queuing
- Horizontal scaling and transactions

The following section describes each of these strategies and techniques.

Simple Failover Using iWay Heartbeat

iWay Service Manager (iSM) can be deployed to automatically fail over to another waiting machine usually referred to as a *hot backup* host. In this model, configuration and repository files are shared so that the backup iSM behavior is identical to the primary iWay Service Manager. Simple failover relies on the native functionality of iWay to emit and respond to *heartbeat* messages which signify normal operation of the primary server. When a failure is detected, the backup host executes a process which manages the switch-over (sending an appropriate message to the router to reconfigure itself, posting an email to the SysOp, and so on) and then assumes the workload of the primary server. It should be noted that the primary and backup servers need not be located in the same data center, for example, they may be geographically dispersed.

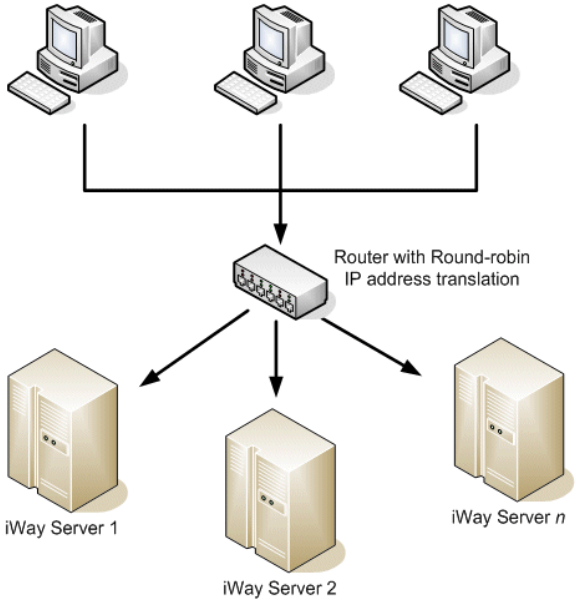


Simple Failover Using Third-Party Tools

A third-party tool clustering or failover product, such as Veritas Cluster Server, can replace the iWay heartbeat, monitoring, and failover process flow logic. In this case, iWay is unaware of the failover management and is run in stand-alone mode. The topology, configuration, and other requirements will be dictated by the needs of the third-party tool.

IP-based Horizontal Scaling

IP traffic is very easy to redirect and load balance, and there are very efficient and robust solutions for managing communications at this level in the protocol stack. Because the content of messages is not inspected, this method of work distribution is extremely fast. Devices such as Cisco 7500 series routers can provide round-robin address translation to distribute requests across several identical iWay Service Managers. In the case of a single server stoppage, the router detects the failure and processing continues on the remaining servers. Sharing of iWay repositories (not shown in the diagram) may also be part of this solution. For maximum reliability, each of the iWay instances can have hot backup failover, implemented either using iWay or a third-party tool.



Web-based Horizontal Scaling

For web traffic (for example, web services, HTTP), a web router can be used to distribute or load balance across the target iWay Service Managers. Stateful transactions can be supported by the use of *session affinity*.

Web-based Horizontal Scaling Using iWay Performance Monitor

iWay Performance Monitor is a web service monitoring and routing solution. Policy-based routing can be used to manage traffic and distribute workloads across iWay instances in complex ways. In a HA environment, the iWay Performance Monitor node should be deployed with hot backup so there is no single point of failure.

Horizontal Scaling for Queuing

An extensive explanation of configuring third-party message queuing products for the HA environment is beyond the scope of this appendix. All mature queuing products support the configuration options needed to scale horizontally without adversely affecting guaranteed, non-duplicated message delivery. The simplest approaches entail allowing multiple consumers to access a queue and message filtering to balance the load between iWay listeners. If that is inappropriate, stateless horizontal scaling can be achieved by using additional instances of iSM and redistributing existing clients to these instances. Stateful horizontal scaling is generally achieved by connecting instances of iSM into a cluster, which allows those instances to communicate with each other, as well as to the application clients.

Horizontal Scaling and Transactions

iWay is optimized for handling stateless processes. Scaling and load balancing may affect the order of processing of messages and may allow a series of related messages to execute on different iWay instances. Because of this, moving to a HA architecture can reveal idiosyncrasies and/or limit design assumptions in the application. Applications that have implied transactions or implied message order dependence may behave differently in the HA environment. Note that iWay is not the source of this changed behavior; any middleware deployed for HA will reveal these types of application flaws. In situations where a web router is part of the iWay HA solution, enabling session affinity may ensure correct application behavior.

iWay Reverse Invocation Proxy and High Availability

The iWay Reverse Invocation Proxy has the ability to distribute transactions to multiple iWay worker instances on other hosts. The workers register themselves with the relay, informing it about which services they (the workers) can provide. Workers may register for mutually exclusive services; workers may register to handle the same services; or workers may do a combination of both, resulting in partially overlapping areas of responsibility.

The proxy is intended for applications where direct connection from the internet/DMZ to the enterprise intranet is not permitted for security reasons. The proxy itself may be horizontally scaled and/or set up to support failover using the mechanisms previously discussed.

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.